POLITEXT

Ernesto J. Cruselles Forner
José Luis Melús Moreno

Secuencias pseudoaleatorias para telecomunicaciones

EDICIONS UPC

Presentación

El empleo de secuencias aleatorias se extiende a un gran número de disciplinas tanto tecnológicas como científicas, existiendo una gran variedad de procedimientos para generarlas. Sin embargo, debido principalmente a los métodos disponibles, las secuencias obtenidas son casi siempre periódicas, con lo que no podrán ser realmente aleatorias y su bondad para ser utilizadas dependerá de cuánto se parezcan a una secuencia verdaderamente aleatoria. Este parecido será una característica determinante de los buenos o malos resultados que se puedan obtener al utilizar las secuencias en diferentes aplicaciones.

La motivación de este libro está, fundamentalmente, en servir de base en los estudios de Ingeniería de Telecomunicación, en especial de Ingeniería Telemática, y en otras muchas disciplinas, tanto técnicas como científicas, en las que este tipo de secuencias son susceptibles de ser aplicadas. También va dirigido a todos aquellos profesionales que en un momento determinado puedan requerir del uso de estas secuencias, tanto si se les plantea el problema de generarlas, como si lo que necesitan es comprobar que aquéllas de las que ya disponen se ajustan a sus requerimientos, sobre todo teniendo en cuenta el gran número de campos en los que se hace uso de las mismas, y que van desde la criptografía a la modelización de redes, el testeo de chips VLSI, la economía, la estadística, la simulación de procesos estocásticos, el análisis de riesgos en sistemas de seguridad, la simulación de fenómenos naturales o las tecnologías de la información (telecomunicaciones , etc.).

El primer objetivo del libro es, pues, presentar la generación de secuencias pseudoaleatorias de forma eminentemente práctica, pero intentando encontrar un equilibrio entre la fácil comprensión para el lector y el rigor formal necesario. Al mismo tiempo también se pretende que el libro sea lo más completo y autocontenido posible.

El segundo objetivo del libro es sintetizar y analizar de forma coherente y adecuada la información referente a este tema, en la actualidad muy diseminada tanto en artículos como en revistas, de tal manera que sea más fácil para el lector introducirse en el tema, y ofrecer además las referencias bibliográficas adecuadas para poder profundizar más si se considera oportuno.

Para conseguir estos dos objetivos se ha dividido el libro en nueve capítulos. En el primer capítulo se hace un estudio de la aleatoriedad y las propiedades que van asociada a ésta, además de definir las propiedades que deben cumplir los dispositivos que se utilicen para generar las secuencias aleatorias. En el segundo capítulo se hace un estudio de los diversos algoritmos de generación de secuencias pseudoaleatorias que se han propuesto para ser empleadas en aplicaciones de *software*, como por ejemplo la modelización y la simulación por ordenador.

Sin embargo, la mayoría de aplicaciones de telecomunicaciones que necesitan usar secuencias pseudoaleatorias las generan mediante procedimientos *hardware*. El capítulo 3 muestra diversas técnicas

para implementar generadores de secuencias pseudoaleatorias, mientras que el capítulo 4 muestra numerosas estructuras que se han propuesto, prácticamente todas ellas en la década de los 80, para generar este tipo de secuencias.

La efectividad en el uso de las secuencias pseudoaleatorias depende en gran medida del método que se haya utilizado para crearlas y que éste sea correcto, es decir, que las secuencias periódicas generadas sean lo más indistinguibles posible de secuencias verdaderamente aleatorias. Por tanto, es fundamental tener un conocimiento amplio sobre sus características y para ello se deben emplear tests estadísticos que permitan detectar desviaciones en el comportamiento de la secuencia respecto al de una secuencia verdaderamente aleatoria. En el capítulo 5 se describen los tests de aleatoriedad más significativos a los que se pueden someter estas secuencias para comprobar su grado de aleatoriedad.

Sin embargo, el hecho de que las secuencias consideradas pasen la mayoría de estos tests estadísticos no garantiza que cumplan la condición de impredictibilidad que necesariamente debe llevar asociada una secuencia verdaderamente aleatoria. En el capítulo 6 se estudiará la complejidad como medida del grado de impredictibilidad, y se verán las medidas de complejidad más utilizadas para estudiar este tipo de secuencias.

En el capítulo 7 se presentan algunas aplicaciones que necesitan usar este tipo de secuencias y los requisitos necesarios de cada una de ellas. Esto puede ser de gran utilidad para, en un momento determinado, permitir al diseñador de estos sistemas obtener de forma eficiente una secuencia que se ajuste a sus necesidades.

En el capítulo 8 se describe un ejemplo práctico de diseño de un generador de secuencias y se dan algunas reglas básicas para obtener ciertas propiedades, y se muestra cómo a partir de una cierta estructura elemental satisfactoria se pueden construir otras más complejas con una adecuada combinación de las mismas.

Sin embargo, y puesto que es casi siempre arduo y difícil estudiar estas propiedades mediante un análisis matemático, es posible hacer una aproximación al problema evaluando su comportamiento respecto a las propiedades descritas anteriormente mediante el uso de técnicas de simulación. Esta comparación se puede llevar a cabo pasando la secuencia de salida del generador por una batería de tests, de forma que obtengamos una comprobación de hasta qué punto cumple las especificaciones deseadas de aleatoriedad. Por ello, se incluye junto al libro un programa informático que, una vez generada una secuencia, permite de forma rápida y eficiente testear su aplicabilidad. En el último capítulo del libro se describe este programa, (instalación, funcionamiento, análisis de los resultados) y se da un ejemplo explicativo.

Nuestro deseo es, pues, brindar al lector un texto que sin rehuir la complejidad y rigurosidad que necesariamente conlleva el estudio de la aleatoriedad, pueda ser leído, en gran parte, con un mínimo bagaje matemático y permita obtener una visión clara y concisa de todos sus aspectos, tanto desde el punto de vista teórico como práctico.

Finalmente, deseamos agradecer de forma muy sincera a Raquel Domingo todos sus desvelos por conseguir componer de forma correcta y amigable este texto.

Indice

1 Aleatoriedad y secuencias pseudoaleatorias

2 Algoritmos de generación por *software*

5. Propiedades estadísticas de las secuencias pseudoaleatorias

6. Complejidad de secuencias

7 Aplicaciones de las secuencias pseudoaleatorias

8 Proceso de diseño de un generador

9 PREDICTEST: Programa de análisis de secuencias pseudoaleatorias

1 Aleatoriedad y secuencias pseudoaleatorias

1.0 Introducción

Hasta hace poco tiempo, los científicos que necesitaban usar números aleatorios en sus trabajos empleaban métodos clásicos y pesados para obtenerlos, como tirar monedas al aire, dados, etc. En 1927, L.C.H. Tippet realizó el primer intento de obtener números aleatorios a gran escala utilizando para ello los resultados obtenidos en varios censos de población, y consiguió 10.400 números aleatorios de cuatro dígitos que presentó de forma ordenada, constituyendo así la primera tabla de números aleatorios [TIP27]. Un esfuerzo similar lo realizaron los españoles Royo y Ferrer en 1954, quienes obtuvieron una tabla de 250.000 dígitos aleatorios a partir de resultados de la Lotería Nacional [ROY54].

Sin embargo, la aparición de las máquinas de cálculo electrónicas trajo consigo el poder generar este tipo de números de forma más eficiente. Un primer intento de generar números aleatorios de esta forma es el que realizaron M.G. Kendall y B. Babington-Smith en 1939, quienes llegaron a generar 100.000 números aleatorios [KEN38].

Otro intento lo realizaron en la compañía Rand Corporation en 1955, y consiguieron una tabla con más de un millón de números aleatorios y 100.000 abscisas aleatorias de la distribución normal tipificada [RAN55], utilizando ya para ello un generador de números aleatorios diseñado para este fín por la propia empresa.

Además, la aparición cada vez más frecuente de sistemas electrónicos para telecomunicaciones que aprovechan las características de este tipo de secuencias para realizar diversos procesos (por ejemplo cifrado, aleatorización, etc.), ha impulsado la aparición de gran número de dispositivos electrónicos, la mayoría de ellos basados en máquinas de estados finitos como los registros de desplazamiento con realimentación lineal, que son capaces de generar este tipo de secuencias para ser utilizadas por dichos sistemas para realizar sus funciones específicas.

Sin embargo, esta forma de generar los números aleatorios presenta una primera cuestión obvia: ¿Cómo pueden considerarse aleatorias las secuencias obtenidas por un procedimiento determinista basado en ejecutar un determinado algoritmo en un ordenador o en un sistema de estados finitos, si de esta forma cada número está completamente determinado por su predecesor?.

La respuesta a esta pregunta es que las secuencias producidas de esta manera no son en realidad aleatorias, sino tan sólo parecen comportarse como tales. Por tanto, a estas secuencias generadas de forma determinista se las llama de forma genérica pseudoaleatorias o cuasi-aleatorias (a veces también pseudoruido debido a la naturaleza aleatoria que presenta el ruido blanco) para indicar que tan sólo

parecen comportarse como verdaderamente aleatorias aunque, evidentemente, no lo son. El problema es que estas secuencias tienden a caer en una rutina, es decir, un ciclo de elementos que acaban repitiéndose, por lo que ya podemos decir que una de las diferencias de estas secuencias con las aleatorias es su periodicidad. Aunque hagamos que el periodo sea tan grande como queramos, lo que siempre será posible actuando sobre algunos parámetros del generador, tarde o temprano la secuencia volverá a repetirse.

La aleatoriedad es una propiedad de un modelo abstracto cuya capacidad para representar o no de forma exacta el comportamiento de la naturaleza es una cuestión filosófica relacionada con el hecho de si el universo es o no determinista y de difícil, por no decir casi imposible explicación, y que va mucho más allá del propósito de este libro, cuyo objetivo es proporcionar unos métodos prácticos para generar secuencias que parezcan tener un comportamiento lo más cercano posible a nuestra noción de aleatoriedad.

Sin embargo, en la naturaleza existen procesos que sí parecen tener un comportamiento caótico, tales como la desintegración radiactiva o el ruido térmico en los transistores, y que podrían aprovecharse para construir generadores de secuencias aleatorias completamente impredictibles para cualquier tipo de aplicación práctica.

Una vez más aparece un problema, que es la enorme dificultad con la que nos encontramos en la práctica a la hora de diseñar circuitos electrónicos que sean capaces de generar secuencias de números aleatorios aprovechando este comportamiento caótico. Esto nos lleva a la necesidad de disponer de una serie de técnicas que, basándose en métodos deterministas sobre los que sí podemos ejercer un control, permitan generar secuencias de números que tengan un comportamiento lo más cercano posible al concepto de aleatoriedad [SCH88][KNU67]. El problema se centrará entonces en hasta qué punto son capaces de conseguirlo.

1.1 Fuentes binarias simétricas

Un generador de números reales aleatorios es un dispositivo cuya secuencia de salida se puede modelar como una secuencia de números estadísticamente independientes y distribuídos según una distribución de probabilidad de tipo rectangular (uniforme) definida en el intervalo [0,1), es decir, que cualquier número dentro de este intervalo debe ser igualmente probable. Esto quiere decir que la fuente simétrica ha ser capaz de generar cualquier secuencia de una cierta longitud con la misma probabilidad. Sin embargo, diseñar circuitos electrónicos que garanticen una independencia estadística y una distribución uniforme de los números generados no es un problema trivial de ingeniería. Por tanto, es fundamental poder testear intensivamente estos generadores para ser capaces de detectar defectos estadísticos, tanto en la fase de su diseño o fabricación como, posteriormente, durante su periodo de operación.

Dentro de las secuencias de números aleatorios, serán de especial interés para nosotros las constituídas por números binarios, es decir, aquéllas cuyos elementos sólo pueden ser el uno o el cero. Ello se debe a que la mayoría de los procesos actuales relacionados con las comunicaciones emplean técnicas digitales basadas en la utilización de estos dos estados.

Por tanto, una fuente simétrica binaria bien diseñada deberá emitir secuencias de una determinada longitud constituídas por ceros y unos, ambos con igual probabilidad. Esto nos lleva a

que será necesario comprobar cuidadosamente que las secuencias que generemos cumplan este objetivo, y para ello es interesante analizar detenidamente el concepto de aleatoriedad. Por ejemplo, si alguien tuviera que decidir si una secuencia determinada como la secuencia todo ceros (000000000000.....) ha sido obtenida tirando monedas al aire, probablemente la respuesta sería que no, ya que intuitivamente se espera que una secuencia verdaderamente aleatoria tenga el mismo número de ceros que de unos. Algo similar ocurrirá con la secuencia (1010101010101010.....) ya que, aunque contiene igual cantidad de ceros que de unos, la consideraríamos también como no aleatoria, debido a que las parejas de bits están distribuídas uniformemente. Problablemente, la secuencia (11001001000011111101101010100...) le parecería aleatoria a la mayoría de la gente. Sin embargo, esta secuencia es simplemente el desarrollo binario del número π y, por tanto, es de una forma muy determinada. Por tanto, el juicio de si una secuencia particular puede ser la salida de un dispositivo que pueda modelarse como una fuente binaria simétrica (y, por tanto, verdaderamente aleatoria) será algo más complejo y dependerá de los diferentes tipos que podamos esperar de no aleatoriedad.

Una aproximación interesante para definir la aleatoriedad de la secuencias generadas mediante algún tipo de algoritmo determinista es la que hace Martin-Löf [MAR66], cuando define de una forma informal la cantidad de aleatoriedad contenida en una secuencia binaria como la longitud del programa más corto (máquina de Turing) que puede generar dicha secuencia. Según esta definición, cualquier secuencia que contenga algún tipo de regularidad, por ejemplo cuando su descripción sea más corta que su longitud, será considerada como no aleatoria.

Sin embargo, la longitud del programa más corto que permite generar la secuencia, y que se llama también la complejidad de Kolmogorov [KOL65], no es computable ni aún disponiendo de recursos de computación ilimitados. En el capítulo 6, que tratará el tema de la complejidad de las secuencias aleatorias, se estudiarán más profundamente estos conceptos.

Esto nos lleva al hecho de que, para estudiar la aleatoriedad de las secuencias producidas por métodos de generación deterministas, deberemos disponer de una serie de herramientas conocidas como tests estadísticos que nos permitan detectar, con una probabilidad elevada, cuándo la secuencia estudiada ha sido generada empleando un modelo estadístico que difiera considerablemente del de una fuente binaria simétrica. En el capítulo 5 se presentan una serie de tests estadísticos que nos permiten comprobarlo.

1.2 Conceptos de aleatoriedad

Antes de continuar con los métodos de generación determinista de secuencias pseudoaleatorias, parece conveniente intentar ir un poco más allá en el concepto de la aleatoriedad y las propiedades que ésta lleva asociadas. A modo de ejemplo, veamos otras dos definiciones de la aleatoriedad de las secuencias dadas por dos autores diferentes.

En 1951, D.H. Lehmer [LEH51] definió una secuencia aleatoria como una noción vaga que incluía la idea de que cada término debía ser impredictible y cuyos dígitos debían ser capaces de pasar un cierto número de tests estadísticos, en función de la aplicación que se le fuese a dar a la secuencia. Una definición más general es la que en 1962 propone J.N. Franklin [FRA63], que define a una secuencia como aleatoria si tiene la propiedad de que es compartida por todas las secuencias infinitas de muestras independientes obtenidas de variables aleatorias con distribución uniforme.

Si bien la definición de Franklin es una generalización de la definición de Lehmer, ya que lo que realmente establece es que las secuencias, para ser aleatorias, deben ser capaces de pasar absolutamente todos los tests estadísticos, esta definición no se puede considerar muy precisa, ya que es muy restrictiva y nos llevaría realmente a considerar que las secuencias aleatorias, sencillamente, no existen.

El hecho de que una secuencia se comporte como si fuese aleatoria puede valer para aplicaciones prácticas, pero plantea una cuestión muy importante tanto desde el punto de vista matemático como filosófico: ¿Qué queremos decir con comportamiento aleatorio?. Es necesario definir de forma cuantitativa este comportamiento, y establecer una serie de propiedades matemáticas que se pueden asociar a este tipo de secuencias según nuestra noción de aleatoriedad. Un estudio de estas características es el que realiza Knuth [KNU67], que se describe brevemente a continuación mediante una serie de definiciones y teoremas.

En primer lugar, se considerarán las secuencias de reales en el intervalo [0,1) de longitud infinita, pero antes se definirá lo que se entiende como secuencia infinita aleatoria, con la siguiente serie de definiciones y propiedades.

Supongamos que hemos generado la secuencia aleatoria de reales entre 0 y 1 de longitud infinita $\{x_n\} = x_0, x_1, x_2,...$, y sean u y v dos números reales que cumplen que $0 \le u \le v \le 1$. Si x es una variable aleatoria que está uniformemente distribuída entre 0 y 1, la probablidad de que $u \le x \le v$ es igual a v-u.

A partir de esta definición ya se pueden establecer las siguientes consideraciones:

DEFINICION 1: La secuencia $x_0, x_1,...$ será equidistribuída (o también 1-distribuída) si, y sólo si, se cumple que $\text{Prob}(u \le x_n \le v) = v - u$, para todo par de números reales u y v tal que $0 \le u \le v \le 1$.

Una generalización natural de la propiedad de equidistribución es considerar parejas de números adyacentes de la secuencia (x_n, x_{n+1}). Para que estas parejas de números también estuvieran uniformemente distribuídas, se requeriría que la secuencia cumpliese la condición:

$$\text{Prob}(u_1 \le x_n \le v_1 \text{ y } u_2 \le x_{n+1} \le v_2) = (v_1 - u_1)(v_2 - u_2)$$

para cualesquiera de los cuatro números u_1, u_2, v_1, v_2 que cumplen que $0 \le u_1 \le v_1 \le 1$ y $0 \le u_2 \le v_2 \le 1$.

Puesto que generalmente obtendremos las secuencias de números reales $R_n = \{r_0, r_1, ...\}$ mediante el empleo de ordenadores que sólo son capaces de generar estos números en un rango entre 0 y m ($0 \le R_n < m, \ \forall n$), podremos obtener a partir de ellos una secuencia de reales $X_n = \{x_0, x_1, ...\}$ en el rango [0,1) mediante la siguiente expresión:

$$x_n = \frac{r_n}{m} \qquad \forall n$$

En general, para cualquier entero positivo k, podemos requerir que nuestra secuencia sea k-distribuída, es decir, que las agrupaciones de k números que hagamos, para cualquier k, también estén uniformemente distribuídas.

DEFINICION 2: Una secuencia será *k*-distribuída si:

$$\text{Prob}(u_1 \le x_n < v_1..., u_k \le x_{n+k-1} < v_k) = (v_1 - u_1)...(v_k - u_k)$$

para cualquier elección de los números reales u_j, v_j con $0 \le u_j \le v_j \le 1$ y $1 \le j \le k$.

Para las secuencias aleatorias de longitud infinita se puede afirmar que si cumplen la propiedad de ser *k*-distribuídas, también serán (*k*-1)-distribuídas, ya que siempre podremos hacer $u_k = 0$ y $v_k = 1$. Por ejemplo, cualquier secuencia que sea 4-distribuída será también 3-distribuída, 2-distribuída y equidistribuída.

La siguiente definición extiende el concepto de *k*-distribución de las secuencias de longitud infinita al límite. La noción de secuencias de números reales ∞-distribuídas, o también llamadas completamente distribuídas, fue introducida por primera vez por J.N. Franklin en 1963 [FRA63].

DEFINICION 3: Se dice que una secuencia es ∞-distribuída si es *k*-distribuída para todos los enteros positivos *k*.

Sin embargo, en 1909 y mucho antes de que Franklin definiera la ∞-distribución de las secuencias aleatorias de números reales, Emil Borel introducía este concepto para secuencias de enteros *b*-arias. Las mismas ideas que se han aplicado a las secuencias de números reales en el intervalo [0,1) se pueden aplicar para las secuencias de números enteros. En particular, consideremos la secuencia *b*-aria $\{S_n\} = S_0, S_1, S_2,...$ formada por un conjunto de números enteros entre 0 y *b* -1. Para el caso binario *b* valdrá 2, es decir, o tendremos un cero o un uno.

Un número *b*-ario $s_1s_2...s_k$, será un conjunto ordenado de *k* enteros donde $0 \le s_n < b$ para $1 \le j \le k$. Podemos redefinir, pues, los conceptos anteriores para secuencias *b*-arias de enteros de longitud infinita.

DEFINICION 4: Una secuencia *b*-aria de longitud infinita será *k*-distribuída si:

$$\text{Prob}(S_n, S_{n+1},..., S_{n+k-1} = s_1s_2...s_k) = (1/b)^k$$

para todos los números *b*-arios $s_1s_2...s_k$. Es decir, que la probabilidad de cualquier subsecuencia de *k* bits será la misma. Por ejemplo, si consideramos las secuencias binarias ($b = 2$) de longitud infinita, diremos que son 2-distribuídas si:

$$\text{Prob}(00) = \text{Prob}(01) = \text{Prob}(10) = \text{Prob}(11) = (1/2)^2 = 1/4$$

Veamos otras definiciones asociadas a las secuencias aleatorias de enteros de longitud infinita.

DEFINICION 5: Una secuencia *b*-aria infinita *k*-distribuída será también (*k*-1)-distribuída, es decir, cumplirá que:

$$\text{Prob}(S_n, S_{n+1},..., S_{n+k-2} = s_1s_2...s_{k-1}) = (1/b)^{k-1}$$

DEFINICION 6: Sea $\{X_n\} = X_0, X_1, X_2,...$ una secuencia de números reales entre [0,1). Si la secuencia:

$$\{\lfloor b_i X_n \rfloor\} = \lfloor b_i X_0 \rfloor, \lfloor b_i X_1 \rfloor, \lfloor b_i X_2 \rfloor \ldots$$

donde $\lfloor \ \rfloor$ indica la parte entera, es una secuencia de enteros b-aria k-distribuída para todo b_i de un conjunto infinito de enteros, $1 < b_1 < b_2 < b_3,...$ entonces la secuencia de reales original $\{X_n\}$ también será k-distribuída.

DEFINICION 7: Si $\{X_n\} = X_0, X_1, X_2,...$ es una secuencia de números reales entre [0,1), que cumple la propiedad de k-distribución, y denotamos como $f(x_1, x_2,..., x_k)$ a una función de k variables integrable por Riemann, entonces

$$\lim_{n\to\infty} \frac{1}{n} \sum_{0\le j<n} f\left(X_j, X_{j+1}, X_{j+k-1}\right) = \int_0^1 \ldots \int_0^1 f\left(x_1, x_2, \ldots, x_k\right) dx_1 \ldots dx_k$$

DEFINICION 8: Una secuencia es (m,k)-distribuída cuando para cada muestreo regular consistente en coger una muestra de cada j $(0 \le j \le m)$, la secuencia que se obtiene también es k-distribuída.

Una secuencia k-distribuída es el caso especial para m =1, es decir, cuando el muestreo consiste en coger todos los elementos. El caso m = 2, que consiste en coger una de cada dos muestras de la secuencia, implica que las k-tuplas en las posiciones pares deben tener la misma densidad que la k-tuplas en posiciones impares.

Por ejemplo, si una secuencia es 3-distribuída, es decir, todas las ocurrencias de 3 números son equiprobables, será (2,3)-distribuída si son 3-distribuídas las secuencias resultantes de tomar un término de cada dos (subsecuencias de números de posiciones pares o subsecuencias de números de posiciones impares). Algunas conclusiones importantes que se derivan de esta definición son:

1) Una secuencia (m,k)-distribuída será (m,μ)- distribuída para $1 \le \mu \le k$.
2) Una secuencia (m,k)-distribuída sera (d,k)-distribuída para todos los divisores d de m.

Las definiciones anteriores nos servirán para definir de forma más estricta las secuencias ∞-distribuídas.

DEFINICION 9: Una secuencia ∞-distribuída será (m,k)-distribuída para todos los enteros positivos m y k.

A partir de las definiciones anteriores ya se pueden obtener una serie de propiedades de las secuencias aleatorias y realizar una formulación válida de la idea intuitiva de aleatoriedad.

PROPIEDAD 1: Una secuencia infinita de reales en el intervalo [0,1) será aleatoria si es ∞-distribuída con probabilidad 1.

PROPIEDAD 2: Una secuencia de enteros *b*-aria infinita será aleatoria si todas las subsecuencias infinitas obtenidas a partir de ella mediante muestreo también son 1-distribuídas (o equidistribuídas).

El muestreo realizado sobre la secuencia no tiene por qué ser regular (el que nos lleva a la definición de secuencia ∞-distribuída), sino que también se pueden considerar muestreos irregulares expresables mediante una función. A modo de ejemplo, supongamos que definimos la regla de muestreo n^2. La secuencia resultante debería ser (s_0, s_1, s_4, s_9,..., s_{n^2},...).

Según la propiedad 2, esta secuencia resultante debería ser equidistribuída si la secuencia que la originó es aleatoria.

De todo lo anterior se puede deducir lo que se considera que debe ser una secuencia de reales infinita aleatoria. Una secuencia infinita de reales $\{X_n\} = X_0, X_1, X_2$,....en el intervalo [0,1) se dice que es aleatoria:

A1) Si es ∞-distribuída.

A2) Si, siempre que sea P una determinada propiedad que haga que $P(\{Y_n\})$ se mantenga con probabilidad 1, para una secuencia $\{Y_n\}$ de muestras independientes de variables aleatorias que siguen una distribución uniforme, entonces $P(\{X_n\})$ también tiene probabilidad 1.

A3) Si toda subsecuencia infinita obtenida a partir de ella también es ∞-distribuída.

A4) Si para cualquier algoritmo que especifique una secuencia infinita de enteros no negativos distintos $\{S_n\}$, para $n \geq 0$, la subsecuencia $X_{s_0}, X_{s_1}, \ldots$ correspondiente a este algoritmo es ∞-distribuída.

A5) Si la secuencia de enteros *b*-aria obtenida a partir de ella mediante la función $\{\lfloor bX_n \rfloor\}$ es aleatoria para todos los enteros $b \geq 2$.

Es conveniente clarificar el punto A4 mediante un ejemplo. Para ello, consideremos la secuencia infinita verdaderamente aleatoria $\{X_n\}$ y, por tanto, ∞-distribuída, según A1. Podemos definir la subsecuencia $\{X_{s_n}\}$ de la siguiente forma: definimos $s_0 = 0$ y s_n ($n > 0$) como el menor entero positivo para el cual los elementos de la subsecuencia $X_{s_n-1}, X_{s_n-2}, \ldots, X_{s_n-n}$ son todos menores que 1/2. La subsecuencia $\{X_{s_n}\}$ también será ∞-distribuída.

Para secuencias *b*-arias infinitas $\{S_n\}$, se puede afirmar que serán aleatorias :

A6) Si toda subsecuencia *b*-aria infinita obtenida a partir de ella mediante muestreo (regular o irregular) es 1-distribuída.

Desde un punto de vista práctico, se podría discutir la utilidad de las secuencias aleatorias infinitas en el sentido que se acaba de describir. Si disponemos de una secuencia binaria ($b = 2$) de la que se sabe que como mínimo es, por decir algún número, 1.000.000-distribuída, se debe tener en cuenta que esta secuencia va a tener subsecuencias de 1.000.000 de ceros y 1.000.000 de unos. Es decir, que las secuencias verdaderamente aleatorias van a mostrar localmente comportamientos no aleatorios. Claramente, esta situación representaría un problema en la mayoría de la aplicaciones.

Por ejemplo, si se dispone de una sistema de cifrado en el que la secuencia cifrada se obtiene combinando (mediante la suma módulo 2) la información que se va a transmitir con dicha secuencia pseudoaleatoria, durante estas subsecuencias de 1.000.000 de ceros (o de unos), la secuencia será enviada al canal sin cifrar.

En una modulación *spread spectrum,* que expande el ancho de banda de la señal a transmitir hasta el ancho de banda del canal, combinándola (mediante suma módulo 2) con la secuencia pseudoaleatoria, el hecho de que aparecieran estas subsecuencias en la secuencia pseudoaleatoria haría que no estuviéramos expandiendo el espectro convenientemente durante toda su duración.

Si bien es cierto que la probabilidad de ocurrencia de este tipo de subsecuencias será tan sólo de $(1/2)^{1000000}$ en promedio según la definición 4, el hecho es que tarde o temprano ocurrirá.

Este problema queda solucionado en la práctica por el hecho de que las secuencias que se van a emplear no serán realmente aleatorias infinitas, sino pseudoaleatorias, obtenidas por máquinas de estados finitos y, por tanto, de naturaleza periódica. El hecho de su periodicidad implicará que no podrán ser ∞-distribuídas.

Así pues, debemos también preguntarnos qué patrón se debe tener en cuenta para definir el comportamiento aleatorio para las secuencias de tipo finito. Se podría argumentar que no hay medio alguno para juzgar legítimamente si una secuencia finita es aleatoria o no, en especial por cuanto cualquier secuencia en particular es tan probable como cualquier otra.

Sin embargo, también para este tipo de secuencias podemos tener una cierta noción de aleatoriedad, puesto que prácticamente todo el mundo coincidiría en considerar a la secuencia finita de longitud 10 (0111010011) como más aleatoria que la secuencia (1010101010), y ésta como más aleatoria que la secuencia (0000000000).

Si bien las secuencias realmente aleatorias (infinitas) mostrarán localmente un comportamiento no aleatorio, esperaremos encontrar también ese comportamiento en una secuencia finita larga, aunque jamás en una corta. Por lo tanto, al igual que ocurría con las secuencias aleatorias infinitas, podemos esperar que presenten el máximo grado de equidistribución posible. Para secuencias finitas se puede afirmar que:

DEFINICION 10: Una secuencia b-aria de longitud N, $\{X_n\} = X_1, X_2,..., X_N$ será k-distribuída si cumple que:

$$\text{Prob}\ (X_n, X_{n+1},..., X_{n+k-1} = x_1x_2...x_k) \approx (1/b)^k$$

para todos los números b-arios $x_1x_2...x_k$.

Para el caso de secuencias binarias, b debe valer 2. Desafortunadamente, mediante esta definición, una secuencia finita puede ser k-distribuída sin ser (k-1)-distribuída, al contrario de lo que ocurría para las secuencias de tipo infinito.

DEFINICION 11: Una secuencia *b*-aria de longitud *N* será aleatoria si es *k*-distribuída para todos los enteros positivos $k \leq \log_b(N)$.

El interés practico consistirá en evaluar la aleatoriedad de las secuencias pseudoaleatorias de longitud finita. Tal como se ha dicho antes, para ello se pueden emplear una serie de tests que nos permitirán comprobar si las secuencias cumplen las propiedades anteriores o no, y detectarán los posibles defectos estadísticos que presenten.

1.3 Postulados de aleatoridad de Golomb

En 1967, S.W. Golomb postuló tres propiedades que se asocian con la aleatoriedad de secuencias finitas [GOL67]. Estos tres postulados, conocidos como los postulados de Golomb, condensan, de una forma sencilla, unas propiedades que cabe esperar en este tipo de secuencias según nuestra intuición de aleatoriedad y nos permitirán realizar, de una forma rápida y práctica, una comprobación de hasta qué punto la secuencia que queremos estudiar cumple con los preceptos de aleatoriedad estudiados en el apartado anterior. Las secuencias que cumplan estos tres postulados se llaman G-aleatorias (o también pseudoruido).

Sin embargo, hay que tener en cuenta que el hecho de que una secuencia sea meramente G-aleatoria no implica necesariamente que sea impredictible (condición que está asociada también a las secuencias verdaderamente aleatorias), pero como mínimo se aseguran unas características aleatorias consideradas buenas. Un ejemplo de esto son las *m*-secuencias, que son producidas por registros de desplazamiento de realimentación lineal (LFSR) y cumplen perfectamente los tres postulados de Golomb.

Tal como se verá en el capítulo 3, donde se estudiará detalladamente este dispositivo, si el registro de desplazamiento con realimentación lineal tiene *L* celdas y el polinomio que representa sus conexiones de realimentación es primitivo, generará una secuencia de periodo 2^L-1. Sin embargo, bastarán conocer 2*L* bits de la secuencia de salida para poder obtener el resto del periodo, empleando para ello el algoritmo de Massey-Berlekamp [MAS76] que se estudiará en el capítulo 6. Por tanto, si bien sí tienen unas propiedades aleatorias buenas ya que cumplen los postulados de Golomb, su impredictibilidad es bastante pobre.

Las secuencias consideradas por Golomb son las secuencias binarias periódicas $\{X_n\} = (x_0, x_1,..., x_{P-1})^\infty$, donde *P* representa el periodo, producidas por los sistemas electrónicos de dos estados (*on* y *off*).

Para describirlas se usan los valores +1 y -1, que se corresponderán con los valores binarios 0 y 1 respectivamente. Esto explica algunas de las fórmulas que se usan para describir los postulados de Golomb y que no se usarían si se utilizara simplemente aritmética módulo 2 (es decir, ceros y unos binarios).

Con esos dos valores, la suma tiene tres valores posibles (2, 0,-2) en lugar de dos (1,0) y el producto es equilibrado, ya que dos combinaciones dan +1 y las otras dos dan -1, en contraste con el caso de usar módulo 2 que está desbalanceado, ya que de las cuatro posibilidades, tres dan suma 0 y sólo una da suma 1.

Una vez aclarado esto, las siguientes propiedades están asociadas con la aleatoriedad:

P1 En cada periodo de la secuencia, la diferencia entre el número de ceros y el número de unos no debe exceder de uno. Esta propiedad se puede expresar de la siguiente forma:

$$\left| \sum_{n=1}^{P} x_n \right| \leq 1$$

P2 En cada periodo de la secuencia, la mitad de las subsecuencias de unos tiene longitud 1, una cuarta parte tiene longitud 2, una octava parte tiene longitud 3, etc. En general, hay $1/2^i$ subsecuencias de longitud i. Lo mismo ocurre con las subsecuencias de ceros. Pero además, debe haber el mismo número total de subsecuencias de ceros que de unos.

P3 La función de autocorrelación normalizada de dicha secuencia $C(\tau)$ debe ser bivalor. Esto se puede poner de forma explícita como sigue:

$$P \cdot C(\tau) = \sum_{n=1}^{P} x_n x_{n+\tau} = \begin{cases} 1 & si \quad \tau = 0 \\ k & si \quad 0 < \tau < P \end{cases} \qquad (1)$$

donde P es el periodo de la secuencia.

Se puede demostrar que $k = -1/P$ si el postulado P1 también se satisface. El que la función de autocorrelación deba ser bivalor tiene un considerable interés en sí mismo. Los fundamentos de autocorrelación de las m-secuencias se verán detalladamente en el capítulo 5. Para ilustrar estos postulados se pueden usar algunos ejemplos simples:

EJEMPLO 1.1: Supongamos una secuencia periódica de periodo $P = 7$ y cuyo primer periodo es : 1110100. Vemos que tiene 4 unos y 3 ceros por lo que cumple con el primer postulado (P1) ya que la diferencia no excede de uno. Si se pasa esta secuencia por el test, en el segundo postulado P2 se obtienen las subsecuencias que muestra la tabla 1.1.

Este segundo postulado establece que del total de subsecuencias de unos (2 en este ejemplo), la mitad deben tener longitud 1, la cuarta parte longitud 2 y así sucesivamente. Lo mismo debe ocurrir para las subsecuencias de ceros. Además, se puede ver en la tabla 1.1 que el número total de subsecuencias de unos y ceros para las longitudes de interés (en este caso, sólo longitud 1) son iguales.

Para el cálculo de la función de autocorrelación de un periodo, se usa la ecuación (1). Para ello, sustituímos cada cero de la secuencia por -1, y dejamos los unos igual, con lo cual queda la secuencia: 1 1 1 -1 1 -1 -1. De esta secuencia se obtiene que:

$$C(\tau) = \begin{cases} -1/7 & si \quad 0 < \tau < 7 \\ 1 & si \quad \tau = 0 \end{cases}$$

que es bivalor tal como se esperaba.

Tabla 1.1 Subsecuencias para el ejemplo 1.1

Subsecuencias / *Longitud*	*Unos*	*Ceros*	*Total (unos+ceros)*
1	1	1	2
2	0	1	1
3	1	0	1
TOTAL	2	2	4

Los tres postulados son independientes, pero cuando se cumple P2 normalmente implica que también es cierto P1. Sin embargo, puede suceder que se cumplan los postulados primero y tercero pero no el segundo, o que se cumplan el primero y el segundo pero no el tercero. Para ilustrarlo, veamos unos ejemplos más.

EJEMPLO 1.2: Sea la secuencia de periodo $P = 11$ $(11011100010)^{\infty}$. Esta secuencia satisface el primer postulado, puesto que se pueden contar 5 ceros y 6 unos, y también el tercer postulado, ya que la autocorrelación sólo tiene dos valores y $k = -1/11$.

Sin embargo no satisface el segundo postulado de Golomb, ya que hay un desequilibrio de dos subsecuencias de ceros por una de unos para longitud 1, tal como se muestra en la tabla 1.2 de subsecuencias.

Tabla 1.2 Subsecuencias para el ejemplo 1.2

Subsecuencias / *Longitud*	*Unos*	*Ceros*	*Total (unos+ceros)*
1	1	2	3
2	1	0	1
3	1	1	2
TOTAL	3	3	6

EJEMPLO 1.3: Sea la secuencia de periodo $P = 13$ $(1111101110010)^{\infty}$. Esta secuencia no cumple ni el primer postulado (tiene 9 unos y sólo 4 ceros) ni el segundo, puesto que vuelve a encontrarse el mismo desequilibrio entre subsecuencias de unos y ceros de longitud 1.

Además, parece fuera de lugar la subsecuencia de unos de longitud 5, aunque los postulados no prevean esta circunstancia. Las subsecuencias correspondientes a este ejemplo se muestran en la tabla 1.3.

Tabla 1.3 Subsecuencias para el ejemplo 1.3

Subsecuencia Longitud	*Unos*	*Ceros*	*Total (unos+ceros)*
1	1	2	3
2	0	1	1
3	1	0	1
4	0	0	0
5	1	0	1
TOTAL	3	3	6

EJEMPLO 1.4: Sea la secuencia de periodo $P = 8$ $(10110010)^{\infty}$. Satisface el primer postulado, ya que hay 4 ceros y 4 unos, y el segundo postulado pero, sin embargo, no muestra una función de autocorrelación bivalor.

El número de subsecuencias de esta secuencia se muestra en la tabla 1.4.

Tabla 1.4 Subsecuencias para el ejemplo 1.4

Subsecuencias Longitud	Unos	*Ceros*	*Total (unos+ceros)*
1	2	2	4
2	1	1	2
3	0	0	0
TOTAL	3	3	6

Por tanto, nuestro interés residirá en ver si se satisfacen estos postulados y, cuando no se cumplan, ver si la desviación es muy grande (por ejemplo, el desbalanceo de unos y ceros en %, por cuántas subsecuencias no se cumple, etc).

Tal como se comentó al comienzo de este apartado, si bien el cumplimiento de estos postulados es fundamental para considerar las secuencias como aleatorias, generalmente se requerirá de ellas, para la mayoría de aplicaciones, que cumplan otra serie de requisitos ya que éstos, por sí solos, no son suficientes para garantizar la impredictibilidad (o también la no-reproducibilidad) de la secuencia.

Esto quiere decir que, si bien las *m*-secuencias producidas por registros de desplazamiento con realimentación lineal cumplen a la perfección los tres postulados de Golomb, y muestran por tanto un comportamiento aleatorio que parece coincidir con el que cumplen las secuencias verdaderamente aleatorias, presentan el problema de que es fácil reproducir toda la secuencia, a partir de tan sólo unos pocos bits de ella, tal como se verá en el capítulo 3.

Evidentemente, ésta es una cuestión contradictoria con el concepto de aleatoriedad, en el cual el conocimiento de porciones de la secuencia no nos debe dar ninguna información útil para poder determinar otras porciones de las cuales no disponemos. Tal como veremos posteriormente, este concepto de impredictibilidad está asociado al concepto de la complejidad de las secuencias.

Sin embargo, el hecho de que las secuencias sean impredictibles es una característica fundamental en ciertas aplicaciones que hacen uso de ellas como la criptografía, (más concretamente el cifrado en flujo que se describirá en el capítulo 7).

Otras aplicaciones de telecomunicaciones como la modulación *spread spectrum*, los sistemas de acceso mútiple por división de código (CDMA), o los sistemas de aleatorización para comunicaciones vía satélite, (que se describirán también en el capítulo 7), no requerirán que se cumpla esta propiedad.

De hecho, todas estas aplicaciones emplean con gran efectividad las secuencias producidas por LFSRs, sin que el hecho de que presenten una elevada predictibilidad sea un inconveniente para su correcto funcionamiento.

Esto recuerda la acertada definición que Lehmer ofrecía de las secuencias aleatorias, en la que la efectividad de dicha secuencia y, por tanto, los requerimientos de los tests que sobre ella se deben realizar para considerarla útil, van a depender en gran medida de la aplicación para la cual la estemos diseñando.

1.4 Requerimientos de los generadores de secuencias pseudoaleatorias

Tal como veremos en los capítulos siguientes, existen un gran número de técnicas para conseguir, con mayor o menor efectividad, que las secuencias pseudoaleatorias generadas presenten una impredictibilidad como la que está inherentemente asociada a las secuencias verdaderamente aleatorias.

Sin embargo, antes de pasar a ver las técnicas más destacables que se han ido proponiendo en la literatura especializada, es conveniente hacer una breve descripción de los requisitos que debe tener un generador de números pseudoaleatorios, si bien éstos se estudiarán con más profundidad más adelante.

Como ya se ha visto, el uso de algoritmos deterministas para generar secuencias aleatorias parece violar el principio básico de aleatoriedad, y por ello, las secuencias producidas por dichos algoritmos deterministas se suelen denominar sintéticas o pseudoaleatorias. Las secuencias producidas deberán mantener ciertos criterios de aleatoriedad, pero siempre comenzarán a partir de un estado inicial (a veces llamado semilla), y producirán los números aleatorios de una manera totalmente determinada y repetitiva.

Las características que se consideran a continuación son generales y deseables para cualquier generador de secuencias pseudoaleatorias, independientemente de la aplicación que se les vaya a dar. Sin embargo, y según cuál sea ésta, pueden aparecer otros criterios adicionales necesarios que en otras aplicaciones pueden no ser restrictivos.

Todos estos criterios se podrán ver en el capítulo 7, donde además se describirán gran número de aplicaciones en las que son necesarios este tipo de secuencias y se destacarán los criterios necesarios para cada una de ellas.

Veamos, pues, cuáles son las propiedades requeridas en toda secuencia pseudoaleatoria:

1) Los números producidos por un generador de secuencias aleatorias deben seguir la distribución uniforme, ya que los sucesos verdaderamente aleatorios siguen esta distribución.

2) Los números producidos deben ser estadísticamente independientes, es decir, el valor que tome un número en una secuencia aleatoria no debe afectar al valor del siguiente número. Este concepto se suele denominar impredictibilidad de las secuencias y se puede generalizar aún más, de forma que conocidos *n* números de la secuencia debe ser muy difícil (idealmente imposible) predecir el siguiente número (*n*+1) con una probabilidad mayor que 1/2.

3) Las secuencias de números pseudoaleatorias producidas deben ser fácilmente reproducibles, para poder repetir el experimento siempre que sea necesario.

4) La secuencia debe ser no repetitiva para una cierta longitud especificada de antemano. La gran diferencia entre las secuencias verdaderamente aleatorias y las que podemos obtener usando los algoritmos que se describirán en capítulos siguientes es que, si bien las primeras son de longitud infinita, las segundas no lo pueden ser debido a los propios métodos que empleamos para generarlas, basados en su mayor parte en máquinas de estados finitos que hacen que sean de carácter periódico. Sin embargo, en la práctica, podemos considerar adecuada la secuencia obtenida mediante este tipo de máquinas si su periodo es de longitud superior a la porción de secuencia que vamos a emplear en nuestra aplicación, es decir, que no vamos a emplear más números de los que tiene el periodo.

5) La generación de números aleatorios debe ser rápida y eficiente, ya que muchas aplicaciones requieren emplear estas secuencias a una elevada velocidad.

6) El método empleado debe presentar la menor complejidad estructural posible. Esto es deseable tanto si la implementación del algoritmo se realiza mediante *hardware* o *software*, si bien en este último caso es importante el uso de la menor cantidad de memoria posible, ya que en la mayoría de los casos suele ser un recurso escaso.

El hecho que las secuencias deban ser reproducibles se convierte en una consideración imprescindible en la mayoría de aplicaciones de ingeniería que van a hacer uso de estas secuencias. Ello es debido a que si un determinado proceso de un sistema debe hacer uso de una secuencia de este tipo para realizar una determinada función, generalmente, en otro punto del sistema, se deberá realizar el proceso inverso, empleando para ello la misma secuencia pseudoaleatoria que empleó el primero. Tal como veremos en el capítulo de aplicaciones, esto hace que no sólo sea de vital importancia la forma de generar las secuencias pseudoaleatorias sino también el hecho de que sea posible generarlas de una forma sincronizada en ambos procesos del sistema. Para este tipo de aplicaciones, que serán la gran mayoría, se pueden hacer varias aproximaciones al problema de tener la misma secuencia en ambos procesos del sistema, que generalmente estarán separados tanto en el espacio (situados en

localizaciones físicas distintas), como en el tiempo (ya que entre el primer proceso y el segundo existirá un lapso de tiempo o retardo que podrá ser variable en función de numerosas consideraciones).

La primera aproximación que se podría hacer al problema consiste en generar la secuencia mediante alguno de los métodos disponibles y almacenarla en un medio físico de almacenamiento, como un disco. Evidentemente, esta técnica presenta numerosos inconvenientes, como por ejemplo la poca versatilidad en el cambio de la secuencia, el elevado coste del medio físico de almacenamiento (sobre todo teniendo en cuenta los enormes periodos que generalmente se requieren en las aplicaciones normales), y el hecho del retardo que implican las lecturas a este tipo de medios. Además, una vez generada la secuencia, si ésta se ha de emplear en diversos puntos del sistema, habría que transportarla de alguna manera a todos ellos, lo que implicaría un gran coste en tiempo y una gran falta de flexibilidad si se requiriese cambiar la secuencia.

Una segunda aproximación al problema consistiría en generar la secuencia mediante algún método y almacenarla en memoria. Si bien esta solución mejoraría el tiempo de acceso a la secuencia respecto al caso anterior, este tipo de memorias tienen un coste mayor que las de tipo masivo. Además, mantendría todos los demás problemas que se presentaban en el caso anterior.

La tercera aproximación, que es la que se usa en la práctica, consiste en emplear un valor de entrada (semilla) a algún algoritmo determinista de forma que, si la semilla es la misma, el algoritmo genere siempre la misma secuencia pseudoaleatoria. De cara a aplicaciones prácticas, esta aproximación soluciona todos los problemas que presentaban las anteriores, ya que en todos los puntos del sistema bastará con que se disponga del algoritmo, bien en forma de programa, bien en forma de dispositivo electrónico, y lo único que se tendrá que enviar a todos los puntos para que generen la misma secuencia pseudoaleatoria será precisamente la semilla (que generalmente será de unas decenas de bits). Una vez considerada esta tercera aproximación como solución práctica para disponer de la misma secuencia pseudoaleatoria en diversos puntos de un determinado sistema, el principal problema que se planteará, y que será motivo de estudio a lo largo de este libro, será el de diseñar algoritmos que sean capaces de generar secuencias que cumplan las propiedades descritas en los puntos anteriores con la mayor sencillez y eficiencia posible.

1.5 Problemas

PROBLEMA 1.1 ¿Puede ser una secuencia periódica equidistribuída?

PROBLEMA 1.2 Considerar la secuencia periódica binaria 0,0,1,1,0,0,1,1,... Determinar si dicha secuencia es 1-distribuída, 2-distribuída y 3-distribuída.

PROBLEMA 1.3 Construir una secuencia periódica ternaria que sea 3-distribuída.

PROBLEMA 1.4 Determinar si la secuencia $(010011011000101)^{\infty}$ cumple los tres postulados de Golomb.

PROBLEMA 1.5 En una secuencia aleatoria de 1000.000 de dígitos decimales, ¿cuál es la probabilidad de que contenga 100.000 de cada uno de los 10 dígitos posibles?

PROBLEMA 1.6 Demostrar que si la secuencia X_0, X_1, ... es k-distribuída, la secuencia Y_1, Y_2,... obtenida mediante

$$Y_n = \lfloor nX_n \rfloor / n$$

también lo es.

PROBLEMA 1.7 Mostrar que la secuencia $\{X_n\}$, definida en el intérvalo [0,1), estará k-distribuída si, y sólo si:

$$\lim_{N\to\infty} \frac{1}{N} \sum_{0\le n\le N} \exp\left(2\pi i\left(c_1X_n+\ldots+c_kX_{n+k-1}\right)\right) = 0$$

para cualquier conjunto de enteros c_1, c_2,...,c_k no todos cero.

PROBLEMA 1.8 Mostrar que la secuencia b-aria $\{X_n\}$ estará k-distribuída si, y sólo si, todas las secuencias $\{c_1X_n+c_2X_{n+1}+\ldots+c_kX_{n+k-1}\}$ estén equidistribuídas, siempre que c_1, c_2,..., c_k sea un conjunto de enteros con mcd(c_1, c_2,..., c_k) = 1.

PROBLEMA 1.9 Mostrar que la secuencia de reales en el intervalo [0,1) $\{X_n\}$ estará k-distribuída si, y sólo si, todas las secuencias $\{c_1X_n+c_2X_{n+1}+\ldots+c_kX_{n+k-1}\}$ estén equidistribuídas, siempre que c_1, c_2,..., c_k sean enteros no todos cero.

PROBLEMA 1.10 Sea X_0, X_1, ...una secuencia binaria ($2k$)-distribuída. Mostrar que:

$$\overline{P_r}\left(X_{2n}=0\right) \le \frac{1}{2} + \binom{2k-1}{k} / 2^{2k}$$

PROBLEMA 1.11 Sea $\{X_n\}$ una secuencia binaria que es aleatoria según la definición A4. Mostrar que la secuencia de reales en el intervalo [0,1) $\{Y_n\}$ definida mediante:

$$\begin{aligned} Y_0 &= 0.X_0 \\ Y_1 &= 0.X_1X_2 \\ Y_2 &= 0.X_3X_4X_5 \\ Y_3 &= 0.X_6X_7X_8X_9 \\ &\ldots\ldots\ldots \end{aligned}$$

también será aleatoria en el sentido de dicha definición.

1.6 Bibliografía

[CHA66] CHAITIN, G. J. *On the Lengh of Programs for Computing Finite Binary Sequences*. Journal of the ACM 13, pp. 547-569, 1966.

[CHA69] CHAITIN, G. J. *On the Lenght of Programs for Computing Finite Binary Sequences: Statistical Considerations*. Journal of the ACM 16, pp.145-159, 1969.

[CHA75] CHAITIN, G. J. *A Theory of Program Size Formally Identical to Information theory*. Journal of the ACM 22, pp. 329-340, 1975.

[CHA87] CHAITIN, G. J. *Information, Randomness and Incompleteness.* World Scientific Publishing, Singapore, 1987.

[FEL68] FELLER, W. *An Introduction to Probability Theory and Its Applications*. 3rd. edn, Vol. 1, New-York, Wiley, 1968.

[FRA63] FRANKLIN, J.N. *Deterministic Simulation of Random Processes*. Math. comp, 17, pp. 28-59, 1963.

[GOL67] GOLOMB, S.W. *Shift Register Sequences*. Holden-Day. 1967.

[KAN84] KANNAN, R.; LENSTRA, A.K.; LOVASZ, L. *Polynomial Factorization and Non-randomness of Bits of Algebraic and Some Trascendental Numbers*. ACM Sysposium on the Theory of Computing, pp.191-200, 1984.

[KEN38] KENDALL,M.G.; BABINGTON-SMITH, B. *Journal of the Royal Statistical Society*. Series A, 101, pp. 147-166, 1938. Series B, 6, pp. 51-61, 1939.

[KKO] K. KO. *A Definition of Infinite Pseudo-Random Sequences*. (Manuscript).

[KNU67] KNUTH, D.E. *The Art of Computer Programming. Vol 2: Seminumerical Algorithms*. Adison-Wesley, 1967.

[KOL65] KOLMOGOROV, A.N. *Problemy Pere Daci Informacii, 1.* pp. 3-11, 1965.

[KOL65] KOLMOGOROV, A.N. *Three Approaches to the Quantitative Definition of Information*. Prob. of Inform. Transmission 1, 1, pp.1-7, 1965.

[LEH51] LEHMER, D.H. *Mathematical Methods in Large-Scale Computing Units*. Annu. Comput. Lab. Harvard Univ. 26, pp. 141-146, 1951.

[MAS76] MASSEY, J.L. *Shift Register Sysnthesis and BCH Decoding*. IEEE Trans on Information Theory, Vol. IT-15, No 1, Jan. 1976.

[MAR66] MARTIN-LÖF, P. *The Definition of Random Sequences.* Information and control, Vol. 9, pp. 602-619, 1966.

[MAU89] MAUER, U.; MASSEY, J. L. *Perfect Local Randomness in Pseudo-random Sequences*. Proceedings Crypto 89, Santa Barbara, 1989.

[MOO40] MOOD, A. M. *The Distribution Theory of Runs.* Ann. Math. Stat., 11, pp 367, 1940.

[POP59] POPPER, K. R. *The Logic of Scientific Discovery*. pp. 162-163, London, 1959.

[RAN55] THE RANDOM CORPORATION. *A Million Random Digits With 100000 Normal Deviates.*The Free Press, New York, Chap. 7, 1955.

[ROY54] ROYO, J.; FERRER, S. *Tablas Auxiliares de Estadística.* Consejo Superior de Investigaciones Científicas, 1954.

[SCH88] SCHORR, C.P. *On the Construction of Random Number Generators and Random Functions Generators*. Proceeding of Eurocrypt´88, Lecture Notes in Computer Science 330, Springer-Verlag, 1988.

[TIP27] TIPPET, L.C.H.. *Random Sampling Numbers.* Cambridge University Press, 1927.

[WOL85] WOLFRAM, S. *Origins for Randomness in Physical Systems.* Physical Review Letters, Vol. 55, pp.449-452, 1985.

2 Algoritmos de generación por *software*

2.0 Introducción

En las últimas décadas se ha desarrollado un gran esfuerzo en la labor de diseñar y testear algoritmos que sean capaces de generar secuencias pseudoaleatorias, debido a la gran cantidad de aplicaciones que usan este tipo de secuencias. Su buen funcionamiento se basa, precisamente, en el grado de aleatoriedad que presenten.

Estos algoritmos difieren no sólo por la técnica empleada, sino también por su velocidad de generación, la longitud de su periodo, la sencillez en su implementación (*hardware* o *software*), y en su parecido con las secuencias verdaderamente aleatorias.

En este capítulo se van a presentar algunos de los algoritmos de generación de secuencias de números aleatorios más conocidos, cuyo diseño y forma de generación está especialmente indicado para ser utilizado en ordenadores, por lo que sus aplicaciones más inmediatas se centrarán en la simulación de procesos, aunque también pueden ser utilizadas para otras aplicaciones, como puede ser el cifrado de las comunicaciones, etc.

2.1 Algoritmos generadores de secuencias pseudoaleatorias

2.1.1 El algoritmo del medio del cuadrado

Este algoritmo, desarrollado a mediados de los 40 por John von Newman, se basa en tomar un número inicial (semilla) y elevarlo al cuadrado, empleando los dígitos del medio para obtener el segundo número de la secuencia. Los siguientes números se van obteniendo iterando este proceso. Veámoslo con un ejemplo:

EJEMPLO 2.1: Se desea generar una secuencia de números de 4 dígitos pseudoaleatorios usando el método del medio del cuadrado. Para ello, se elige como semilla el número $x_0 = 3187$:

$$(3187)^2 = 10156969 \quad x_1 = 1569$$
$$(1569)^2 = 02461761 \quad x_2 = 4617$$
$$(4617)^2 = 21316689 \quad x_3 = 3166$$

$(3166)^2 = 10023556 \quad x_4 = 0235$
$(0235)^2 = 00055225 \quad x_5 = 0552$
$(0552)^2 = 00304704 \quad x_6 = 3047$
$(3047)^2 = 09284209 \quad x_7 = 2842$

Siguiendo con este proceso se irían obteniendo los números 0769, 5913, 9635, 8332, 4222, 8252,.....

Sin embargo, las secuencias obtenidas mediante esta técnica presentan gran número de problemas como bajo periodo (es decir, que suelen caer rápidamente en una rutina que se repite continuadamente) y el hecho de que si en algún momento se produce el número cero, todos los números siguientes de la secuencia serán el cero.

Si bien este último problema se puede presentar en la mayoría de los generadores de secuencias pseudoaleatorias, la mayoría de las veces éstos se podrán diseñar de forma que se garantice que el número cero no se va a producir. En este generador es imposible garantizar esto. El siguiente ejemplo muestra este caso.

EJEMPLO 2.2: Se desea generar una secuencia de números de dos dígitos pseudoaleatorios empleando el método del medio del cuadrado, partiendo de la semilla $x_0 = 44$:

$(44)^2 = 1936 \quad x_1 = 93$
$(93)^2 = 8649 \quad x_2 = 64$
$(64)^2 = 4096 \quad x_3 = 09$
$(09)^2 = 0081 \quad x_4 = 08$
$(08)^2 = 0064 \quad x_5 = 06$
$(06)^2 = 0036 \quad x_6 = 03$
$(03)^2 = 0009 \quad x_7 = 00$
$(00)^2 = 0000 \quad x_8 = 00$

A pesar de la gran sencillez de este generador, es recomendable evitar su uso debido a las deficiencias que presenta recurriendo a otros generadores que los solucionan.

2.1.2 El generador del sistema GPSS/360

El GPSS es un lenguaje de propósito general para simulaciones propuesto por primera vez por Gordon en 1961 [GOR61], con el propósito de servir como herramienta para la simulación de sistemas discretos. Posteriormente han ido apareciendo nuevas versiones como la GPSS V o la GPSS/360, que incorpora 8 generadores de secuencias pseudoaleatorias que se demominan RN-1, RN-2,...,RN-8 [FEL68].

El funcionamiento de estos algoritmos es el siguiente: en primer lugar, se requieren 3 vectores de 8 celdas, 8 vectores base que almacenarán las semillas, 8 vectores que contengan los multiplicadores de los generadores (inicialmente se pondrán a 1) y 8 vectores que contengan los índices para cada generador (y que inicialmente estarán todos puestos a 0). Cada número aleatorio se generará mediante el siguiente algoritmo:

Paso 1	La palabra apropiada del vector de índice se hace apuntar al vector de base que contiene la semilla. Puesto que cada índice está inicialmente puesto a 0, el primer número base se emplea como semilla de todos los generadores.
Paso 2	El contenido del vector multiplicador apropiado se multiplica por el contenido del vector base elegido en el paso 1.
Paso 3	Si el bit de mayor peso de los 32 bits de menor orden de este producto es un 1, estos 32 bits de menor peso se sustituyen por su complementario.
Paso 4	Los 31 bits resultantes de este posible producto transformado se almacenan en el vector multiplicador para un uso futuro.
Paso 5	En el vector de índice se almacenan 3 bits de los 16 bits de mayor orden del producto obtenido en el paso 2 para un uso futuro.
Paso 6a	Si se requiere un número fracionario, el número constituído por los 32 bits del medio del producto se divide por 10^6 y el resto que se obtiene es el número de 6 dígitos deseado.
Paso 6b	Si se requiere un entero, los 32 bits del medio del producto obtenido se dividen por 10^3 y el resto producido es el número deseado de 3 dígitos.

Este generador comienza operando como si se tratara de un generador congruencial multiplicativo (que se describe en el punto 2.1.4.A1) pero, a diferencia de él, no emplea una relación de recursión para obtener los sucesivos números pseudoaleatorios. Si bien la motivación de este algoritmo fue inicialmente la de reducir la no aleatoriedad de las secuencias de números obtenidas, el método empleado hace difícil su análisis.

2.1.3 El generador PRN

Este generador se basa en una técnica muy simple para generar números reales pseudoaleatorios entre 0 y 1 y que consiste en la siguiente recursión:

$$x_{n+1} = \lceil 10^b m x_n \rceil$$

donde $\lceil p \rceil$ denota la parte fraccional del número real p, b es el número de dígitos del número pseudoaleatorio y m es una constante multiplicadora que cumple que $0 < m < 1$.

Los estudios realizados sobre las secuencias de salida de este generador han llevado a considerar que un valor de m igual a $10^{-b}(200X \pm Y)$, donde X es cualquier entero no negativo, e Y es cualquier número del conjunto {3, 11, 13, 19, 21, 27, 29, 37, 53, 59, 61, 67, 69, 77, 83, 91}, ofrece buenos resultados para la secuencia pseudoaleatoria de salida. Además, la semilla se debe elegir de la forma $x_0 = 10^{-b}k$, donde k debe ser cualquier entero no divisible por 2 ó 5 y en el rango $0 < k < 10^b$. Veamos el funcionamiento de este generador mediante un ejemplo:

EJEMPLO 2.3: Sea la semilla $x_0 = 0{,}33$, $X = 0$ e $Y = 11$. Queremos obtener una secuencia de números pseudoaleatorios de 2 dígitos empleando el método PRN. Para ello hacemos que $m = 10^{-2}(11{,}0) = 0{,}11$, con lo que la secuencia se producirá de la siguiente forma:

$$
\begin{aligned}
x_0 &= 0,33 \\
x_1 &= \lceil 100(0,11)(0,33) \rceil = 0,63 \\
x_2 &= \lceil 100(0,11)(0,63) \rceil = 0,93 \\
x_3 &= \lceil 100(0,11)(0,93) \rceil = 0,23 \\
x_4 &= \lceil 100(0,11)(0,23) \rceil = 0,53 \\
x_5 &= \lceil 100(0,11)(0,53) \rceil = 0,83
\end{aligned}
$$

La principal desventaja de este generador radica en su lentitud para producir los números pseudoaleatorios, ya que se requieren muchas multiplicaciones para generar cada uno de ellos.

2.1.4 Generadores de congruencias

Los generadores congruenciales han encontrado gran difusión como generadores de números aleatorios en ordenadores. En las librerías de casi todos los ordenadores se puede encontrar una función que implementa esta técnica para generar números aleatorios. Su enorme éxito se encuentra en su sencillez y velocidad de ejecución. Sin embargo, en muchos casos, las secuencias producidas por estos generadores presentan defectos estadísticos, y en algunos casos realmente importantes.Este hecho, y dependiendo de su grado, puede no ser decisivo en algunas aplicaciones donde priman las características de sencillez y rapidez, pero en otras aplicaciones, como por ejemplo la criptografía, puede ser crítico.

A) Generadores de congruencias lineales

Veamos dos tipos de generadores congruenciales lineales, los multiplicativos y los mixtos, y sus propiedades en función de los valores que seleccionemos para los parámetros que los definen.

A1) Generadores congruenciales lineales multiplicativos

En 1951, D.H. Lehmer [SCH79] propuso un algoritmo que ha sido ampliamente estudiado desde entonces. La definición del algoritmo implica únicamente dos parámetros: el módulo m, que debe ser un

número entero primo de elevado valor, y el multiplicador *a*, que debe ser un entero con un rango entre 2 y *m*-1. La generación de la secuencia de enteros s_0, s_1, s_2, s_3,... se produce mediante la siguiente ecuación iterativa:

$$s_{n+1} = f(s_n) \qquad \text{para } n = 1, 2,...$$
$$f(s) = as \bmod m$$

donde la función generadora $f(\cdot)$ se define para todo $s \in \{1, 2, \ldots, m-1\}$.

El generador se debe inicializar eligiendo una semilla s_0 del conjunto $\{1, 2, \ldots, m-1\}$. Como paso adicional, y puesto que generalmente se requieren números pseudoaleatorios reales pertenecientes al intervalo [0,1), la secuencia se puede normalizar a este intervalo mediante su división por el módulo, y producir la secuencia, x_0, x_1, x_2, ..., donde:

$$x_n = \frac{s_n}{m} \qquad \text{para } n = 1, 2, ...$$

A un generador de este tipo se le suele denominar generador congruencial lineal multiplicativo de módulo primo, y representa un buen ejemplo de elegancia y simplicidad [PAY69]. El hecho de que el módulo *m* sea primo evita que la secuencia de salida se colapse en el número cero ya que $f(s) \neq 0$ $\forall s \in \{1, 2, \ldots, m-1\}$. Además, en este generador los valores 0 y 1 son imposibles de conseguir (en el caso de la secuencia normalizada), y los valores máximo y mínimo que se pueden obtener para los números de la secuencia x_n son 1/*m* y -1/*m* respectivamente. Otra cuestión que se debe tener en cuenta es que la normalización no afecta al hecho de que la secuencia parezca aleatoria o no.

Aunque la secuencia de salida de este algoritmo presenta algunos defectos estadísticos, si los parámetros de salida se eligen adecuadamente y si la implementación *software* que se realiza es correcta, produce una secuencia de periodo elevadísimo de números que satisfacen prácticamente todos los tests de aleatoriedad. Sin embargo, otro factor que se debe tener en cuenta en la elección de los parámetros *a* y *m* es que la secuencia de salida presente periodo máximo ($P_{max} = m-1$). Veamos con un ejemplo un caso de generador que produce una secuencia de periodo máximo.

EJEMPLO 2.4: Sea el algoritmo $f(s_{n+1}) = 6s_n \bmod 13$. Si se elige como semilla $s_0 = 1$, la secuencia que se produce es 1, 6, 10, 8, 9, 2, 12, 7, 3, 5, 4, 11, 1, ..., que como se puede ver es periódica, de periodo máximo $P = m-1 = 12$. El hecho de que la secuencia sea de periodo máximo es importante debido a que en este caso la secuencia obtenida será independiente de la semilla elegida. Al variar ésta, la salida será siempre la misma pero desplazada circularmente, y cada uno de los números entre 1 y *m*-1 aparecerá con total seguridad (y una sola vez) en cada periodo.

Una elección comúnmente aceptada de los parámetros es $a = 16807$ y *m* el primo de Mersenne $2^{31}-1$ [FISHM]. El multiplicador $a = 7^5 = 16807$ fue sugerido por primera vez por Lewis, Goodman y Miller en 1969 [LEW69] que basaron la elección en que $f(s_{n+1}) = 16807s_n \bmod 2147483647$ [WIC82] es una función generadora de periodo máximo y probadas propiedades estadísticas. Sin embargo, su implementación en un lenguaje de alto nivel no es trivial, ya que existe la posibilidad de desbordamiento

debido a dicho producto. Para estos valores, se necesitan al menos 46 bits para representar el máximo valor que puede tomar el producto *as*. Sin embargo, el algoritmo de Lehmer es altamente portable entre distintos ordenadores, y puede implementarse correctamente en la mayoría de ellos. Algunos paquetes de *software* comerciales que incluyen este algoritmo como subrutina en sus librerías son la librería IMSL (subrutina RNUN) y el lenguaje de simulación SLAM (subrutina DRAND).

La aparición de programadores más preocupados por la optimización de códigos que en la generación de números aleatorios hizo que se desarrollaran generadores multiplicativos de módulo no primo de la forma $m = 2^b$, donde b es el tamaño de la palabra entera del ordenador. Esto conlleva una mayor eficiencia de ejecución aunque, como contrapartida, las implementaciones obtenidas son muy poco portables entre distintas máquinas, presentan graves defectos estadísticos y además no permiten obtener el periodo máximo. De hecho, el máximo periodo que puede obtenerse es sólo $2^{b-2} = m/4$, y se obtendrá si, y sólo si, (a mod 8) vale 3 ó 5, y la semilla es un entero impar. Si se cumplen estas condiciones, el generador producirá una permutación periódica de la mitad de los enteros impares entre 1 y 2^{b-1}. El ejemplo más representativo lo muestra la función RANDU del IBM/360, que implementa la función $f(s_{n+1})=65539s_n \bmod 2^{31}$, que fue introducida a principios de los 60 y que diversas pruebas han demostrado su pésima calidad. Afortunadamente, a mediados de los 60, la cuestión de la velocidad de ejecución dejó de tener la importancia vital que presentaba hasta entonces y la elección de un módulo primo volvió a coger auge, al menos entre los especialistas. Esto, añadido a la aparición de la aritmética de 32 bits como estándar para ordenadores, hizo que $2^{31}-1 = 2147483647$ reapareciera como una elección lógica para el módulo.

A2) Generadores congruenciales lineales mixtos

Los generadores congruenciales lineales mixtos [GOR71] son una generalización de los anteriores, añadiendo un parámetro aditivo b que satisface que $b \bmod m \neq 0$. Veamos cuál es el algoritmo para generar los números aleatorios. Para ello, sean a, b y m tres enteros positivos con $m>\max\{a,b\}$. Para cada entero no negativo $x<m$, se define una secuencia x_i tal que:

$$x_{n+1} = \begin{cases} x & si \quad n = 0 \\ (ax_n + b) \bmod m & si \quad n > 0 \end{cases}$$

La salida de este generador es la secuencia finita periódica $x_0, x_1, ..., x_{m-1}$.

EJEMPLO 2.5: Sea $f(x_{n+1}) = (5x_n+3) \bmod 16$, con la semilla $x_0 = 1$. La secuencia producida por este generador será: {1, 8, 11, 10, 5, 12, 15, 14, 9, 0, 3, 2, 15, 4, 7, 6, 1, 8, 11, ...} que tiene periodo máximo m-1 = 15.

Al añadir este parámetro aditivo, esta función es capaz de generar el número 0 ($x_0 = 0$) y por lo tanto, una secuencia de periodo máximo será aquélla cuyo periodo sea m y no m-1 como antes. Teóricamente, el módulo m puede ser cualquier entero positivo, sea primo o no, aunque en la práctica se

suele tomar de forma que sea una potencia de 2 o de 10. En el caso de que se tome un módulo del tipo $m = 2^z$, el generador mixto sólo será de periodo máximo si, y sólo si, a mod 4 = 1 y b es impar [GOR71].

Ejemplos de este generador son la subrutina RAND del sistema operativo UNIX, que implementa la función $f(x_{n+1}) = (1103515245x_n+12345) \bmod 2^{31}$, o el generador de números aleatorios del Turbo Pascal estándar que implementa la función $f(x_{n+1}) = (129x_n+907633385) \bmod 2^{32}$, que es de periodo máximo y con el que se obtiene una salida normalizada dividiendo por 2^{32} (en el Turbo Pascal 87 se divide por 2^{31} y si el resultado u es mayor que 1, la salida es 2-u). Se ha demostrado que ambos generadores no son excesivamente buenos en cuanto a aleatoriedad. En el primero de ellos, los bits de menor peso de los números generados no son muy aleatorios, mientras que el segundo también presenta el problema de tener ciclos de pequeño periodo en los bits de menor peso, además de presentar un multiplicador demasiado pequeño (aunque esto produce simplicidad en la representación binaria).

En cualquier caso, veamos algunas consideraciones prácticas necesarias a la hora de diseñar un generador de estas características. Tal como se ha mencionado antes, es necesario realizar una buena elección de los parámetros a, b, m y la semilla x_0 para obtener periodo máximo y buenas propiedades estadísticas.

Estas condiciones también serán válidas para el generador de congruencias lineales multiplicativo visto en el apartado anterior, aunque en ese caso el parámetro b valdrá cero. Veamos estas condiciones, expuestas por Knuth [KNU67]:

a) Elección del módulo *m:* En primer lugar hay que tener en cuenta que el valor del módulo debe ser lo mayor posible, ya que el periodo que obtengamos será siempre menor o igual que él. Además, se debe elegir un valor que facilite el cálculo de la solución de la relación congruencial. Para máquinas que utilicen una representación binaria, una excelente elección es 2^k-1, donde k es la longitud de palabra de la máquina.

b) Elección del multiplicador a y la constante aditiva b*:* Un generador de estas características tendrá el máximo periodo ($P = m$) si, y sólo si:

(1) b es primo relativo con m (mcd(b,m) = 1).
(2) a -1 es un múltiplo de todo primo que divide a m.
(3) a -1 es un múltiplo de 4 si m es un múltiplo de 4.

Estas restricciones nos llevan a que el multiplicador a debe tomar la forma $a = p^v+1$, donde p es la base usada en la representación de los números en el ordenador, y k es la longitud de la palabra del ordenador (número de bits por palabra), $m = p^k$ y $p \le v < k$. En particular, y tal como se vio en el generador congruencial multiplicativo, las elecciones $a = 2^{16}+5 = 65.541$ ó $a = 2^{16}+3 = 65.539$ han mostrado dar resultados satisfactorios. Para la elección del parametro aditivo b basta con que cumpla la condición de que sea relativamente primo con el módulo m.

c) Elección de la semilla x_0*:* Si los parámetros anteriores se han elegido de forma que se obtenga un generador de periodo máximo ($P = m$), la elección de la semilla es irrelevante, ya que el generador producirá con cualquiera de ellas la secuencia completa. Sin embargo, si se usa un generador congruencial

multiplicativo se debe tener cuidado en no elegir la semilla $x_0 = 0$, ya que produciría la secuencia de salida de generador todo ceros. Se puede concluir sobre los generadores congruenciales lineales mixtos que no son una elección adecuada para generar números aleatorios, ya que con una parte suficientemente larga de la secuencia se pueden obtener los parámetros que definen el generador: *a*, *b* y el módulo *m*.

A3) El generador congruencial lineal aditivo

Veamos otro generador congruencial que consigue mayores velocidades de generación que los anteriores, ya que no requiere realizar multiplicaciones. El generador congruencial aditivo, propuesto por Green, Smith y Klem a finales de los 50, requiere de una semilla constituída por *k* números x_1, x_2,..., x_k.

La obtención de números pseudoaleatorios se realiza mediante la aplicación de un algoritmo que produzca una extensión de este número hacia la secuencia x_{k+1}, x_{k+2}, x_{k+3},...., empleando para ello la fórmula recursiva:

$$x_{n+1} = (x_n + x_{n-k+1}) \quad mod \quad m$$

Con este generador se pueden conseguir secuencias de números pseudoaleatorios mayores que *m* (a diferencia de lo que ocurría en los congruenciales lineales multiplicaticos y mixtos, en los que el máximo periodo que se podía obtener era precisamente el módulo *m*) y con buena aleatoriedad. Sin embargo, el comportamiento teórico de este generador no es tan conocido como el de los congruenciales mixtos y, por tanto, hay que tener mucho cuidado si se elige este método para generar números pseudoaleatorios en validarlo previamente (el libro adjunta un programa que puede servir como herramienta para validar las secuencias producidas por generadores pseudoaleatorios, tal como se describirá más adelante). Veamos un ejemplo para comprender mejor el funcionamiento de este generador.

EJEMPLO 2.6: Extendemos la secuencia 1, 2, 4, 8, 6 ($k = 5$) empleando el algoritmo congruencial aditivo y empleando un módulo $m = 10$.

$x_1 = 1$
$x_2 = 2$
$x_3 = 4$
$x_4 = 8$
$x_5 = 6$
$x_6 = (x_5+x_1) \bmod 10 = (6+1) \bmod 10 = 7$
$x_7 = (x_6+x_2) \bmod 10 = (7+2) \bmod 10 = 9$
$x_8 = (x_7+x_3) \bmod 10 = (9+4) \bmod 10 = 3$
$x_9 = (x_8+x_4) \bmod 10 = (3+8) \bmod 10 = 1$
$x_{10} = (x_9+x_5) \bmod 10 = (1+6) \bmod 10 = 7$
$x_{11} = (x_{10}+x_6) \bmod 10 = (7+7) \bmod 10 = 4$
$x_{12} = (x_{11}+x_7) \bmod 10 = (4+9) \bmod 10 = 3$
$x_{13} = (x_{12}+x_8) \bmod 10 = (3+3) \bmod 10 = 6$

$x_{14} = (x_{13}+x_9) \bmod 10 = (6+1) \bmod 10 = 7$
$x_{15} = (x_{14}+x_{10}) \bmod 10 = (7+7) \bmod 10 = 4$
$x_{16} = (x_{15}+x_{11}) \bmod 10 = (4+4) \bmod 10 = 8$
$x_{17} = (x_{16}+x_{12}) \bmod 10 = (8+3) \bmod 10 = 1$
$x_{18} = (x_{17}+x_{13}) \bmod 10 = (1+6) \bmod 10 = 7$
$x_{19} = (x_{18}+x_{14}) \bmod 10 = (7+7) \bmod 10 = 4$
$x_{20} = (x_{19}+x_{15}) \bmod 10 = (4+4) \bmod 10 = 8$

B) El generador congruencial cuadrático

El método congruencial cuadrático es especialmente útil para generar secuencias pseudoaleatorias cuando el módulo *m* es una potencia de 2 y representa un método útil para conseguir secuencias con mayor aleatoriedad que con los métodos anteriores. Este método se basa en la siguiente fórmula recursiva:

$$x_{n+1} = (dx_n^2+ax_n+b) \bmod m \qquad n \geq 0$$

Un algoritmo congruencial cuadrático de gran interés es el propuesto por R.R. Coveyou en [COV60] cuando el módulo *m* es una potencia de 2. Para ello usa la relación:

$$x_0 \bmod 4 = 2$$
$$x_{n+1} = x_n(x_n+1) \bmod 2^e$$

Este método es casi equivalente al método del medio del cuadrado para precisión doble visto en el punto 2.1.1, pero las secuencias generadas son de mayor periodo y presentan buenas propiedades estadísticas. Además, este generador se puede implementar de forma eficiente sin problemas de desbordamiento. Veamos su funcionamiento mediante un ejemplo.

EJEMPLO 2.7: Generemos una secuencia pseudoaleatoria usando un generador congruencial cuadrático con parámetro $d = 1$, $a = 1$ y $b = 0$ y con módulo $m = 16$ y semilla $x_0 = 2$.

$x_0 = 2$
$x_1 = (2(3)) \bmod 16 = 6$
$x_2 = (6(7)) \bmod 16 = 10$
$x_3 = (10(11)) \bmod 16 = 14$
$x_4 = (14(15)) \bmod 16 = 2$
$x_5 = (2(3)) \bmod 16 = 6$
$x_6 = (6(7)) \bmod 16 = 10$
$x_7 = 14$
$x_8 = 2$

Podemos observar que en este ejemplo el periodo obtenido es 4, que es mucho menor que el módulo m=16.

2.1.5 Generador basado en el algoritmo de MacLaren-Marsaglia

Este algoritmo [MAC65] constituye un excelente generador de números aleatorios a partir de dos secuencias pseudoaleatorias $\{X_n\}$ e $\{Y_n\}$ producidas por cualquiera de los métodos vistos anteriormente. La forma de generación que se decribe a continuación es muy recomendable debido a los excelentes resultados que presentan las secuencias producidas. Si los periodos de las secuencias $\{X_n\}$ e $\{Y_n\}$ son relativamente primos, producirán una secuencia de periodo muy elevado y con una dependencia muy pequeña entre términos cercanos. El hecho de que este método de generación sea mejor que los vistos hasta ahora es debido a que que sabemos que las secuencias de partida $\{X_n\}$ e $\{Y_n\}$ ya son bastante aleatorias y, por tanto, se reduce considerablemente la posibilidad de obtener degeneraciones. Para su implementación se requiere del uso de un vector auxiliar V[0], V[1], ..., V[k-1], donde k es un número elegido por conveniencia generalmente en las proximidades de 100. Inicialmente, este vector se llenará con los k primeros valores de la secuencia $\{X_n\}$. El algoritmo de generación es el siguiente:

Paso 1	[Generar X e Y]. Actualizar X e Y en los siguientes valores de las secuencias $\{X_n\}$ e $\{Y_n\}$.
Paso 2	[Extraer j]. Actualizar $j \leftarrow \lfloor kY / m \rfloor$, donde m es el módulo empleado en la secuencia $\{Y_n\}$. Por tanto, j es un valor aleatorio determinado por Y tal que $0 \le j < k$.
Paso 3	Tomar como número aleatorio de salida V[j] y actualizar V[j] $\leftarrow$ X.
Paso 4	Volver al paso 1 para generar el siguiente número.

Se puede concluir sobre este generador que satisface virtualmente cualquiera de los requisitos de aleatoriedad para una secuencia generada mediante un ordenador. Además, el tiempo requerido de generación de esta secuencia es sólo ligeramente mayor que el doble del tiempo requerido para generar la secuencia $\{X_n\}$ sola.

2.1.6 Generador de secuencias pseudoaleatorias basado en permutaciones

Veamos un método propuesto por S.G. Akl y H. Meiger [AKLME] que se basa en la realización de permutaciones para generar secuencias de números aleatorios y que es apropiado para ser implementado por s*oftware* debido a su estructura algorítmica.

Los conceptos básicos referentes a este generador son los siguientes: Sea S un conjunto de números binarios con cardinalidad $\#S$ y sean $\bullet$ y o dos operaciones binarias definidas sobre este conjunto. Se puede construir una secuencia $\{S_i\} = s_0, s_1, s_2,...$ de la siguiente forma:

FOR i = 1,2,.... DO
Paso 1: Sea x_0 un elemento arbitrario del conjunto S
Paso 2: $x_i = x_{i-1} \bullet q$, para i >0 y para algún $q \in S$.
Paso 3: $s_i = x_{k_i} \circ x_{k_{i+1}} \circ \ldots \circ x_{k_{i+k-1}}$ para todo $i \geq 0$ y algún entero positivo k.
END FOR

El elemento q se debe elegir de forma que la secuencia $\{x_i\}$ definida en los pasos 1 y 2 tenga un periodo elevado. En el mejor de los casos, la secuencia x_i tendrá un periodo igual a $\#S$, lo que implicará que el periodo T de la secuencia $\{S_i\}$ sea:

$$T = \frac{\#S}{mcd(k, \#S)}$$

En la práctica, hacer que $\#S > 2^{200}$ garantizará que la secuencia no se repita para cualquier aplicación en la que podamos emplear el generador. Veamos cómo se puede emplear el algoritmo anterior para construir una secuencia de permutaciones pseudoaleatorias. Si llamamos P al conjunto de todas las permutaciones posibles en un conjunto de enteros Z_n: (0, 1, 2,..., n-1), se puede calcular la secuencia pseudoaleatoria de permutaciones $\{P_i\}$ como:

FOR i = 1,2,.... DO
Paso 1 : Sea x_0 un elemento arbitrario de Z_n y $q \in Z_n$ de forma que mcd(n,q) = 1.
Paso 2: $x_i = (x_{i-1}+q) \bmod n$, para $i > 0$.
Paso 3: $p_i = f(x_{k_i}) \circ f(x_{k_{i+1}}) \circ \ldots \circ f(x_{k_{i+k+1}})$ para todo k, $i \geq 0$ y algún entero positivo k.
END DO

donde $\circ$ representa la composición de permutaciones. La función f representa una aplicación entre el conjunto Z_n y el conjunto de permutaciones P: $f: Z_n \rightarrow P$. Knott [KNO76] y Knuth [KNU81] muestran dos formas distintas de realizar esta aplicación, aunque aquí consideraremos la que establece Knuth. La secuencia generada en el paso 2 es un caso especial de las secuencias congruenciales lineales que se vieron en el punto 2.1.4A de este capítulo ($x_i = (ax_{i-1}+q) \bmod m$) con $a = 1$ y $m = n$. Eligiendo q mutuamente primo con n se garantiza que la secuencia $\{x_i\}$ tenga periodo máximo n. Si llamamos $p \in P$ a la permutación definida por la secuencia (p(0), p(1),..., p(n-1)), la función f que aplica Z_n en P se puede definir como [KNU81]:

FUNCTION f(x):
  $p(i)$ = i, para i = 0, ..., n-1
  FOR $i = n$-1, n-2,..., 1, 0 DO
  Intercambia ($p(i)$, $p(x_i)$)
  END FOR
RETURN f

Se pueden encontrar otras formas de definir la función f, como la que muestra Lehmer [LEH51]. A continuación se muestra un algoritmo eficiente para implementar un generador de secuencias pseudoaleatorias a partir de la secuencia pseudoaleatoria de permutaciones. Para ello, sea b_i un vector de n bits arbitrario y sea $p[b_i]$ un vector de bits que se obtiene como resultado de aplicar una permutación p al vector b_i. En el algoritmo que se describe a continuación emplearemos una secuencia $\{b_i\}$ definida como:

a) $b_0 = (01010...0101)$
b) $b_i = p_{i-1}[b_{i-1}]$, para $i > 0$

El algoritmo aplica las permutaciones $p_i = f(x_{2i}) \circ f(x_{2i+1})$ a los vectores b_i pero no lo hace calculando p_i, sino aplicando x_{2i} y x_{2i+1} directamente a b_i.

```
n  ←  64
k  ←   2
x0, x1,..., xn-1   ←  Valor inicial
q0, q1,..., qn-1   ←  Entero mutuamente primo con n
b0, b1,..., bn-1   ←  0,1,0,1,...,0,1
REPEAT (tantas veces como se requiera)
        DO  k  TIMES
                {Cálculo de x+q mod n!}
                Acarreo  ←   0
                FOR  i =  1, 2,..., n-1   DO
                        xi  ←  xi  +  qi  +  Acarreo
                        IF   xi > i  THEN   xi ← xi-(i+1)
                                            Acarreo  ←  1
                                    ELSE    Acarreo  ←  0
                END  FOR
                {Aplica  x  a  b}
                FOR  i = n-1, n-2,..., 1   DO
                        Intercambiar (bi, bxi )
                END   FOR
        END DO
        OUTPUT   b0, b1, ..., bn-1
END REPEAT
```

En el algoritmo se ha tomado $k = 2$ y $n = 64$. Con estos valores, el periodo de la secuencia $\{x_i\}$ y, por tanto, de $\{P_i[b_i]\}$ será aproximadamente de 2^{300}.

Implementando este algoritmo basado en permutaciones en lenguaje C y ejecutándolo sobre un ordenador VAX 11/780, es capaz de generar una secuencia de bits pseudoaleatorios a una velocidad de 10.000 bits por segundo.

La secuencia producida por este generador muestra muy buenos resultados al ser analizada mediante los tests estadísticos que se describen en el capítulo 5. Esto, junto con el hecho de que presente máximo periodo, hacen que este generador sea de gran utilidad para cualquier aplicación que necesite usar este tipo de secuencias.

2.1.7 Generador basado en el atractor de Hénon

Veamos un generador de secuencias pseudoaleatorias basado en una aplicación cuadrática bidimensional desarrollada por M. Hénon en 1976 [HEN76], y cuyas propiedades todavía no se conocen completamente [FOR91]. Esta aplicación, que se realiza sobre los puntos del plano, tiene una forma muy simple, definida por $(x_{i+1}, y_{i+1}) = H(x_i, y_i)$ de la siguient foma:

$$x_{i+1} = y_i + 1 - 1.4x_i^2$$
$$y_{i+1} = 0.3x_i$$

Dependiendo del punto inicial (x_0,y_0) que se elija, la secuencia de puntos obtenida mediante la iteración de *H*, o bien divergerá hacia el infinito, o bien convergerá hacia un atractor consistente en un conjunto de puntos del plano.

Hénon demuestra además en [HEN76] que si este punto inicial se elige en el cuadrilátero con vértices A(-1,33, 0,42), B(1,32, 0,133), C(1,245, -0,14) y D(-1,06, -0,5), todos los puntos obtenidos de la iteración caerán dentro de una región contenida por completo dentro de este cuadrilátero.

Una propiedad interesante de la aplicación *H*, es que es extremadamente sensible al punto inicial elegido. La secuencia de puntos obtenida empleando el vector inicial $(x_0 + \Delta x, y_0 + \Delta x)$ estará prácticamente incorrelada con otra secuencia de puntos cuyo vector inical hubiera sido (x_0,y_0). Esta propiedad puede ser muy interesante para una aplicación como la criptografía, ya que este punto inicial podría constituir la clave del criptosistema.

Sin embargo, para aplicaciones de comunicaciones, la secuencia de puntos obtenida será de mayor utilidad si se transforma en una secuencia binaria. Si bien existen infinitas formas de transformar los números reales en dígitos binarios, aquí se mostrará una que hace que la secuencia binaria tenga aproximadamente el mismo número de ceros que de unos. Para ello buscamos un valor real x_m tal que $\text{Prob}(x_i \leq x_m) = \text{Prob}(x_i > x_m) = 0{,}5$. R. Forré [FOR91] realiza un estudio sobre 237 secuencias de Hénon de longitud 1.000 con valores iniciales elegidos aleatoriamente, y concluye que el valor $\hat{x}_m = 0{,}39912$ es una buena estimación de x_m. Con esto, ya se puede definir una forma de obtener la secuencia binaria pseudoaleatoria $\{s_i\}$ mediante:

$$s_i = \begin{cases} 0 & si \quad x_i \leq \hat{x}_m \\ 1 & si \quad x_i > \hat{x}_m \end{cases}$$

donde x_i es la coordenada horizontal del *i*-ésimo punto de la secuencia de Hénon. Esta secuencia estará más o menos balanceada. También se podría hacer esta transformación comparando las coordenadas

verticales y_i con su valor medio $\hat{y}_m = 0,11977$, o bien comparar alguna combinación lineal ax_i+by_i con su valor medio estimado.

Las secuencias producidas por este generador muestran un buen comportamiento aleatorio, puesto que presentan funciones de autocorrelación sin picos para puntos distintos de cero, presentan distribuciones de *n*-tuplas muy satisfactorias (para *n* pequeños), un comportamiento respecto al perfil de complejidad lineal muy bueno, y una entropía por bit muy cercana a 1. Estos conceptos (que se estudiarán en el capítulo 5) pertenecen a medidas que se pueden realizar sobre las secuencias generadas para estimar su grado de aleatoriedad. No se puede afirmar rotundamente que las secuencias producidas por este generador sean verdaderamente aleatorias. Sin embargo, no es posible determinar el tamaño de la memoria de estas secuencias. Dicho tamaño sería *M* si un símbolo de la secuencia dependiera de *M* símbolos previos a ella. Por tanto, la secuencia real de Hénon tendrá memoria 1, ya que cada símbolo producido depende solamente del punto anterior. Este resultado es heurístico, y no es posible por tanto afirmar con absoluta certeza que la secuencia de Hénon no se repetirá al cabo de un cierto tiempo. Sin embargo, el hecho de que existan infinitos puntos en el atractor nos puede hacer pensar que nunca obtendremos exactamente el mismo punto dos veces. Además, el hecho de tener errores de redondeo debidos a la limitación de la representación del ordenador empleado favorecerá que no se produzca este suceso, ya de por sí improbable.

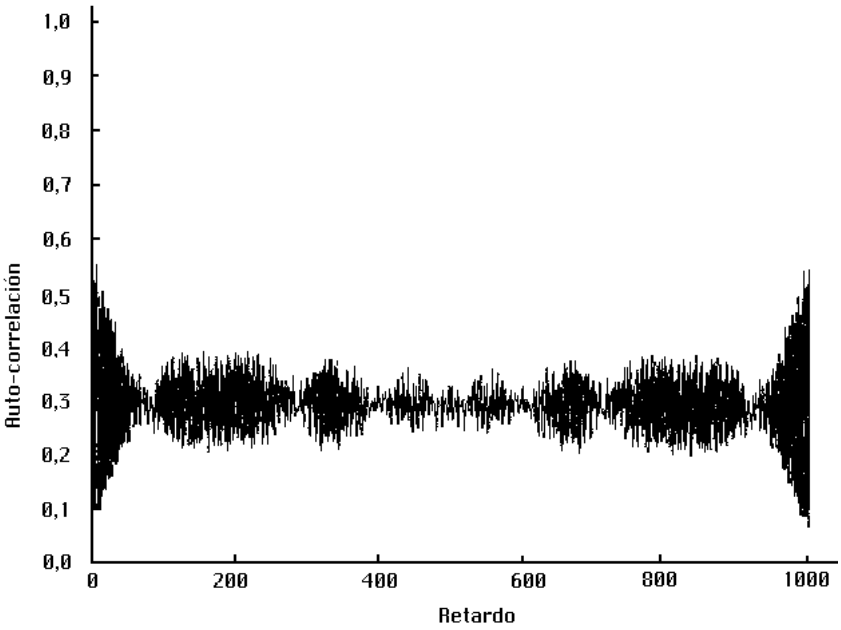
2.1.8 Generador de secuencias caóticas

Este generador, propuesto por M. Romera, I. Jiménez y J. Negrillo [ROM90], está basado también en la generación de secuencias pseudoaleatorias mediante el empleo de funciones caóticas. Si bien el análisis matemático de las órbitas caóticas que rigen su comportamiento no lineal es muy complejo, se puede cuantificar la bondad de la secuencia pseudoaleatoria resultante utilizando los test de aleatoriedad que se describirán más adelante.

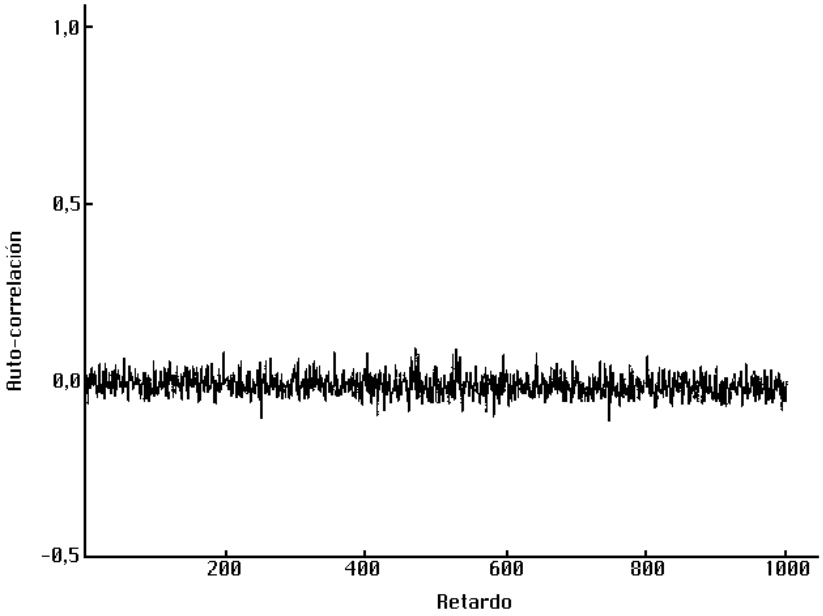
Dentro de lo que constituye la teoría de fractales, el conjunto de Mandelbrot se define como un conjunto de puntos $c = c_{real}+i\,c_{imag}$ del plano complejo, de forma que al iterarse según la expresión $z_{n+1} = z_n^2+k$ $(n = 0, 1, 2,..., z_0 = 0+0i)$ permanecen acotados. Si k es real (puntos del eje del conjunto), veamos un método para generar una secuencia de números aleatorios basado en esta iteración de Mandelbrot:

$x_0 \leftarrow 0$
$k \leftarrow$ Un valor entre -2 y 0,25
FOR $n = 0, 1, 2,...$ DO
$x_{n+1} = x_n^2+k$
END FOR

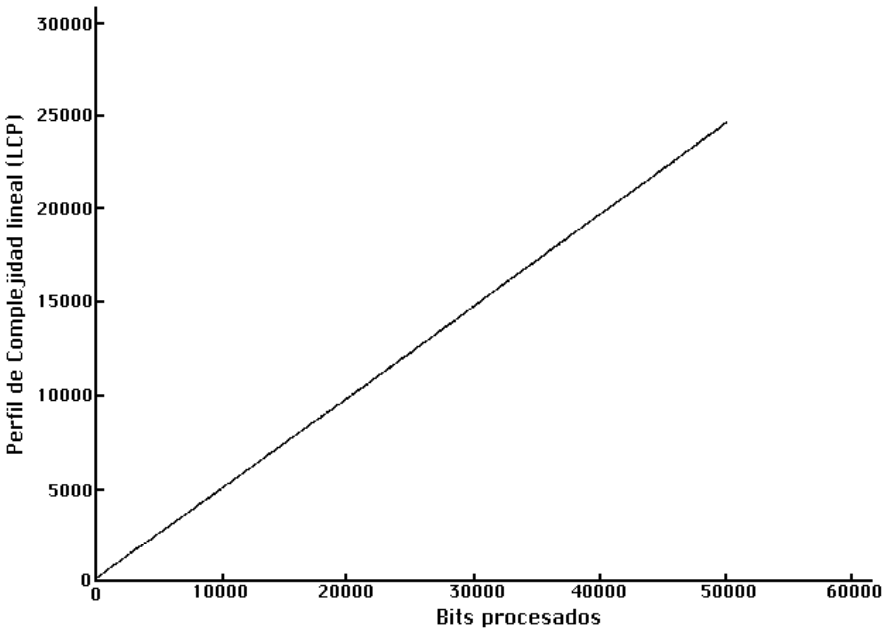
Los números que se obtienen a partir de esta expresión, están acotados en el intervalo [-2, 2] para valores de k en el intervalo $-2 \leq k \leq 0,25$, pero unos son cíclicamente convergentes (periodos 1, 2, 3 ...), y otros son caóticos. Cuando k es muy próximo a -2, por ejemplo k = -1,9999XXXXXXXX, las series que se obtienen a partir de la iteración real de Mandelbrot son casi siempre caóticas.



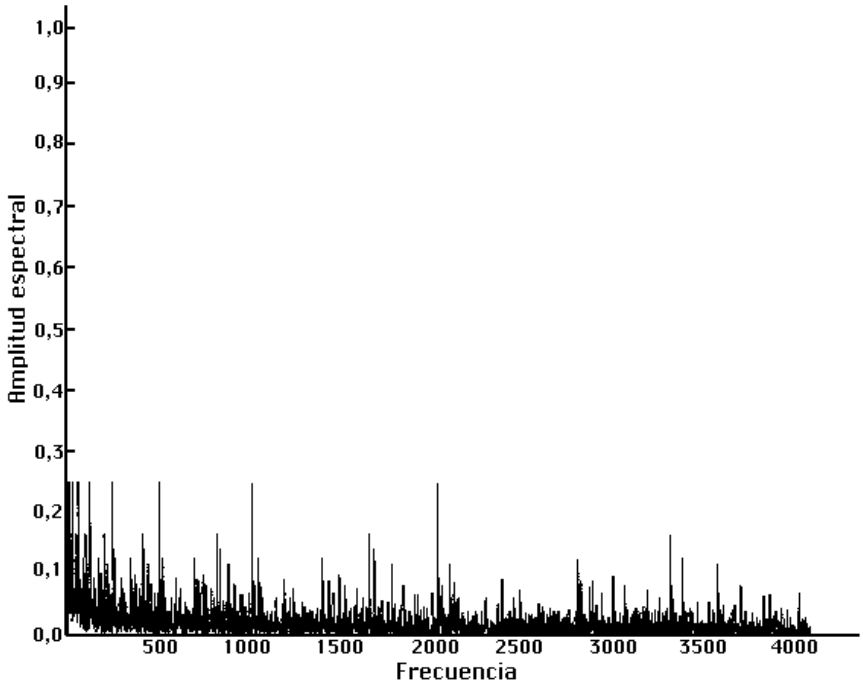
Gráfica 2.1 Autocorrelación para k=-1,935399



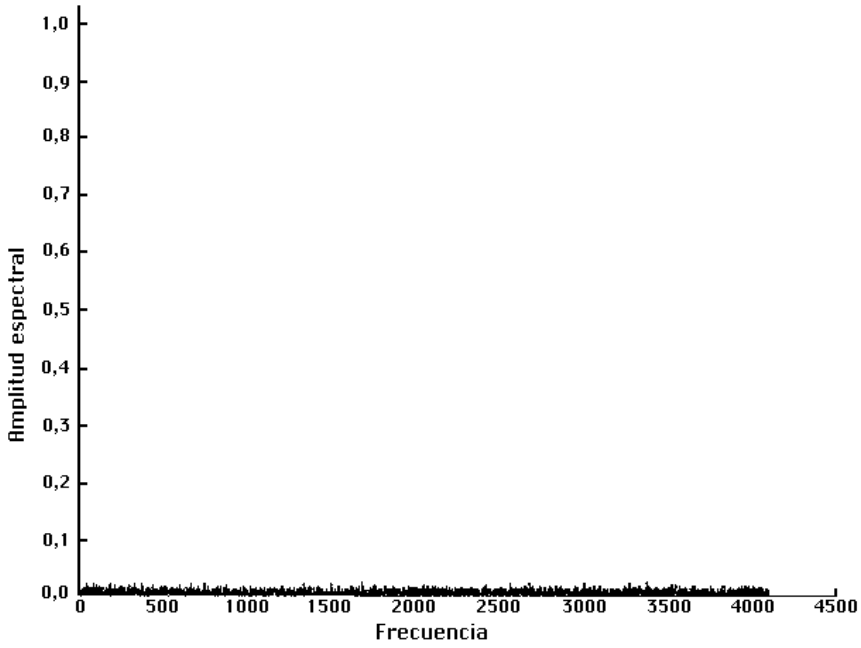
Gráfica 2.2 Autocorrelación para k =- 1,999991234



Gráfica 2.3 Perfil de complejidad para una secuencia con semilla k = -1,999999999999



Gráfica 2.4 Test espectral para k = -1,9



Gráfica 2.5 Test espectral para k =- 1,9999123456

La secuencia real obtenida puede convertirse fácilmente en una secuencia pseudoaleatoria binaria, tomando el signo de cada número de la serie (la distribución es simétrica respecto a cero), o aplicando un criterio de paridad a los dígitos de cada número de la serie. Al igual que el generador anterior, también basado en funciones caóticas, no se puede afirmar con rotundidad que las secuencias producidas no tengan periodo.

Por tanto, es un generador de secuencias aleatorias y no pseudoaleatorias, y algunos tests que se pueden utilizar para determinar la bondad de las secuencias aleatorias, como los postulados de Golomb, o la complejidad y el perfil de complejidad lineal (que se verán en detalle en capítulos siguientes) no son totalmente válidos, ya que se han de pasar sobre un periodo completo de la secuencia. Sin embargo, sí se pueden usar como medidas orientativas de la bondad del generador.

Si se eligen valores de la semilla por debajo de -1,95 para este generador, se obtienen unos valores muy pobres en cuanto a la autocorrelación de las secuencias obtenidas. Sólo a partir de estos valores se obtendrán resultados aceptables, que además mejorarán considerablemente a medida que se vaya acercando al valor -2 (-1,9999....).

A modo de ejemplo, se han estudiado dos secuencias de 50.000 bits producidas por este generador, la primera de ellas, con una semilla k = -1,935399, y la segunda, con una semilla k = -1,9999991234.

Para la primera secuencia, se han obtenido 18.626 ceros y 31.374 unos, mientras que para la segunda se han obtenido 24.763 ceros y 25.237 unos. Se observa claramente que el número de ceros y unos está mucho más equilibrado en la segunda secuencia.

Las gráficas 2.1 y 2.2 muestran el comportamiento de la función de autocorrelación para dos secuencias de 1.000 bits con k = -1,935399 y para k = -1,9999991234 respectivamente, donde se puede ver la evolución de la función de autocorrelación de este generador para estas dos semillas respectivamente. De la comparación de dichas gráficas, se observa que a medida que la semilla se aproxima a -2 desaparecen por completo los picos fuera de fase. Tal como establecía el tercer postulado de Golomb, la existencia de estos picos en la función de autocorrelación era un claro indicativo de dependencias estadísticas entre los bits de las secuencias que eran incompatibles con un buen comportamiento aleatorio.

Otra característica interesante de este generador es la gran sensibilidad que presentan las secuencias de salida frente a variaciones pequeñas de la semilla empleada. Esto es muy interesante para aplicaciones como la simulación o la criptografía, ya que en éstas puede ser interesante disponer de varias fuentes de datos aleatorios que sean lo más independientes entre ellas. El valor de k puede además constituir la clave secreta de un algoritmo criptográfico y el número de claves posibles diferentes dependerá pues, del número de dígitos que el ordenador o sistema generador sea capaz de manejar.

También es necesario estudiar el comportamiento estadístico de las secuencias producidas por este generador, que es muy bueno si se emplean semillas más cercanas a -2 que a -1,9999, y que se refleja en el resultado positivo que dan las secuencias obtenidas al ser sometidas a los diversos tests estadísticos.

Una medida de interés que se puede realizar sobre las secuencias pseudoaleatorias es su complejidad lineal. Si bien esta característica se estudiará en profundidad en el capítulo 6, podemos avanzar que su comportamiento será un indicativo de la impredictibilidad de la secuencia. Para una secuencia verdaderamente aleatoria, dicha complejidad debe crecer (aunque irregularmente) siguiendo una línea de pendiente $n/2$, donde n es el número de bits que se van procesando.

La gráfica 2.3 muestra el buen comportamiento respecto al crecimiento de la complejidad lineal de una secuencia de 50.000 bits obtenida a partir de este generador empleando la semilla k = -1,999999999999.

Otra medida interesante de la aleatoriedad de las secuencias es la que proporciona el test espectral que se verá en el capítulo 5, y se basa en calcular la transformada discreta de Fourier (DFT) de la secuencia sometida a estudio. Para secuencias verdaderamente aleatorias, el resultado obtenido fruto de aplicar este test no debe tener otro pico que no sea el del origen. Las gráficas 2.4 y 2.5 muestran los resultados de aplicar este test sobre dos secuencias de 4.000 bits obtenidas con las semillas k = -1,9 y k = -1,9999123456 respectivamente. Una vez más, se puede observar la evolución positiva a medida que la semilla se aproxima a -2.

Podemos concluir del estudio de este generador que si se usa una semilla k tal que $-2 < k < -1{,}9999$, se obtienen excelentes resultados de la secuencia de salida que, además, no es periódica. Sin embargo, de cara a aplicaciones prácticas presenta el problema de un reducido espacio de semillas.

2.1.9 Generador 1/p

Este generador se basa en la transformación de estado $F(x_{i+1}) = bx_i \bmod m$ de un generador congruencial lineal con base b (b >1).

Para el generador 1/p, el módulo p será un entero p mutuamente primo con b y la semilla un número entero x_0 elegido aleatoriamente entre 0 y p ($x_0 \in Z_p$). Básicamente, lo que hace el generador 1/p

es expandir la semilla x_0/p a la base b, es decir, la secuencia de cocientes b-arios resultante de dividir x_0 por p en base b.

El generador congruencial, a diferencia del aquí presentado, producía una secuencia b-aria de residuos resultantes de dividir x_0 por p en base b. Veamos cómo produce las secuencias pseudoaleatorias este generador. Los parámetros de entrada serán la base b >1, un entero p tal que mcd(p,b) = 1 y un número entero elegido al azar $x_0 \in Z_p$. El algoritmo es el siguiente:

FOR i = 0, 1, 2... DO

Paso 1 Calcular la función del siguiente estado: $x_i = F(x_i - 1) = bx_{i-1} \mod p$

Paso 2 Calcular la función de salida: $z_i = f(x_{i-1}) = bx_{i-1} \ div \ p$

END FOR

El periodo T de la secuencia producida por este generador puede valer como máximo $T = p$-1 cuando p es primo y b es una raíz primitiva módulo p. Las secuencias producidas por este generador serán secuencias de De Bruijn generalizadas de periodo p -1.

Estas secuencias, que se estudiarán al final del siguiente capítulo, tienen la propiedad de que cada subsecuencia b-aria de $|p| - 1$ dígitos aparece por lo menos una vez, y cada subsecuencia b-aria de $|p|$ dígitos aparecerá como mucho una vez en un periodo de la secuencia ($|p|$ denota la longitud de la expansión b-aria de p). Por lo tanto, las secuencias producidas por este generador tienen un gran parecido con las de máxima longitud producidas por los LFSR (m-secuencias), por lo que habrá que tener cuidado en no utilizarlas en aplicaciones como las de cifrado debido a que, si bien presentan un buen comportamiento estadístico, son bastante predictibles.

2.1.10 Generador basado en la suma entera de secuencias binarias

El empleo de funciones que combinen diversas secuencias pseudoaleatorias es una práctica muy común para diseñar generadores cuyas secuencias de salida presenten mejores propiedades, tal como se verá en el siguiente capítulo. Sin embargo, puesto que la suma de enteros es una operación básica de un ordenador, vamos a ver un generador que se basa en emplearla como función combinadora de otras dos secuencias pseudoaleatorias. Podemos suponer que las secuencias de entrada a esta función combinadora son las expansiones binarias de secuencias de reales pseudoaleatorias obtenidas mediante cualquiera de los métodos anteriores, o bien secuencias pseudoaleatorias obtenidas directamente en forma binaria mediante cualquiera de los métodos que se describen en el capítulo 4. Definimos la función suma de enteros f: $Z^N \rightarrow Z$ como:

$$z = \sum_{i=1}^{N} x_i$$

Esta función se puede calcular en forma de una serie de bits comenzando desde el bit menos significativo. Veamos un generador basado en esta función. Para ello, consideremos que las entradas de esta función combinadora son las secuencias producidas por N registros de despazamiento con realimentación lineal con L_j número de celdas respectivamente. El algoritmo de generación será:

FOR i = 1, 2,.... DO

Paso 1 Dar un paso a cada LFSR para producir N bits $x_{1i}, x_{2i}, \ldots, x_{Ni}$

Paso 2 Calcular la suma de los N bits anteriores como si fueran enteros: $S_i = \sum_{k=1}^{N} x_{ki} + C_{i-1}$

Paso 3 Establecer la salida z_i y el bit de acarreo C_i: $z_i = S_i \bmod 2$

$$C_i = \left\lfloor \frac{S_i}{2} \right\rfloor$$

END FOR

En este algoritmo, C_i representa el bit de acarreo de la suma y $\lfloor b \rfloor$ la parte entera del número real b. La suma de N secuencias periódicas será periódica con periodo P:

$$P = \prod_{i=1}^{N} P_i$$

si los periodos P_i de las secuencias individuales que se suman son mutuamente primos entre sí dos a dos. Además, si se suman dos m-secuencias binarias obtenidas a partir de dos registros de desplazamiento con realimentación lineal con L_1 y L_2 celdas respectivamente que son mutuamente primos entre sí, la complejidad lineal de la secuencia obtenida será muy próxima a su periodo y estará acotada superiormente por:

$$\Lambda(z) \leq (2^{L_1} - 1)(2^{L_2} - 1)$$

En cuanto a la distribución estadística, si las secuencias de entrada son independientes y están uniformemente distribuídas, la secuencia de salida también constará de variables aleatorias independientes y uniformemente distribuídas.

Además, la suma de enteros da inmunidad a la correlación de orden (N-1), que es la máxima que se puede obtener, y que implica que la salida estará incorrelada con N-1 de las secuencias que sirven como entrada. Esta propiedad se verá más a fondo en el siguiente capítulo.

Este generador es interesante para aplicaciones como el cifrado en flujo debido a que la suma de enteros presenta una elevada no linealidad, además de dar una elevada inmunidad a la correlación. En este tipo de aplicación, la clave (que es como se denomina a la semilla en aplicaciones criptográficas) puede estar constituída por los contenidos iniciales de las celdas de los N LFSR y el bit de acarreo.

Si bien se presenta este generador para ser implementado usando un ordenador, en el capítulo 4 se mostrará también una versión para ser implementada utilizando elementos *hardware*.

2.1.11 El generador de mochila

Este generador aprovecha la no linealidad de la transformación de la mochila para generar secuencias pseudoaleatorias. El problema de la mochila consiste en, dado un conjunto finito de L pesos enteros positivos y un entero positivo S, ver si existe un subconjunto de esos pesos que sumen exactamente S.

Para un conjunto dado de pesos, la mochila puede verse como una función entre un vector $x = x_1,..., x_L$ y un entero S que es la suma de los pesos seleccionados por x. Veamos cómo realizar el generador de secuencias pseudoaleatorias a partir de esto.

Como elementos de partida vamos a tener un LFSR con L celdas (modulo Q), una mochila de L pesos $w_1,..., w_L$ cada uno de ellos de N bits y el vector inicial del LFSR $x_0 = x_{10},\ldots,x_{L0}$. El algoritmo de generación será el siguiente:

FOR $i = 1, 2,...$ DO

Paso 1 Dar un desplazamiento al LFSR para obtener el siguiente estado.

Paso 2 Calcular la suma de la mochilla: $S_i = \sum_{k=1}^{L} x_{ki} w_i \quad mod \quad Q$

Paso 3 Extraer algunos bits de S_i para formar la secuencia de salida Z_i.

END FOR

La secuencia obtenida mediante la suma de la mochilla será periódica de periodo 2^L-1. Esto implica que la j-ésima secuencia de bits $\{S_j\}$ será también periódica de periodo 2^L-1. Además, si $Q = 2^N$, la complejidad lineal Λ de esta secuencia estará acotada mediante las siguientes expresiones:

$$\Lambda(S_j) = \begin{cases} \leq \sum_{k=1}^{2^j} \binom{L}{k} & j < \lfloor \log L \rfloor \\ \leq \sum_{k=1}^{L} \binom{L}{k} = 2^L - 1 & j \geq \lfloor \log L \rfloor \end{cases}$$

2.1.12 El generador de Blum-Micali

Este generador, definido por M. Blum y S. Micali [BLU82] se basa en el problema de calcular el logaritmo discreto. Sea p un número primo y sea α un elemento generador para el anillo de unidades

módulo p Z_p^*. Se define el logaritmo discreto de un elemento $y \in Z_p^*$ con respecto a α como el único entero ($0 \le x \le p-2$), tal que $y = \alpha^x \mod p$, y lo vamos a denotar como $indice_{p,\alpha}(y)$. El problema del logaritmo discreto con entrada p, α, y consiste en encontrar $indice_{p,\alpha}(y)$, y no existe ningún algoritmo eficiente para solucionar este problema. Veamos cómo obtener un generador de secuencias pseudoaleatorias a partir de él. Para ello se define el siguiente predicado binario:

$$mitad_p(x) = \begin{cases} 1 & si \quad x < \dfrac{p-1}{2} \\ 0 & en \quad otro \quad caso \end{cases}$$

Las entradas para este generador serán los parámetros (p, α) y una semilla $x_i \in Z_p^*$ elegida al azar, y el algoritmo de generación será:

FOR i = 1,...., l DO

Paso 1 $z_i = mitad_p(x_i)$

Paso 2 $x_{i+1} \leftarrow \alpha^{x_i} \mod p$

END FOR

La salida de este generador serán los números z_1, z_2,... Las secuencias generadas de este modo parecen especialmente interesantes para aplicaciones criptográficas, ya que están basadas en la dificultad de calcular logaritmos discretos módulo p con lo que garantizarán la seguridad. En esta aplicación, la clave consistirá en la semilla x_1.

2.2 Generación de números aleatorios siguiendo distribuciones determinadas

Si bien los métodos de generación de números aleatorios, como el del medio del cuadrado o los métodos congruenciales, producen números que siguen una distribución uniforme estándar, puede ser que nos interese obtener secuencias de números pseudoaleatorios que sigan otros tipos de distribución, como por ejemplo distribuciones exponenciales, gausianas (o normales), de Poisson o gamma, que se emplean más a menudo que la distribución uniforme en otras aplicaciones, por ejemplo para realizar simulaciones por ordenador.

Todas estas técnicas se basan en obtener uno o más números aleatorios a partir de la distribución uniforme, y luego aplicar sobre ellos una transformación que los lleve a cumplir la distribución no uniforme deseada.

La generación de números aleatorios empleando métodos de muestreo se suele denominar técnica de Monte-Carlo, y muestra un valor inestimable cuando el modelo que se pretende estudiar presenta una complejidad analítica que lo hace inabordable o muy difícil de tratar.

Veamos a continuación un método de gran utilidad para realizar esta transformación entre una distribución uniforme y otra cualquiera.

2.2.1 El método de la transformación inversa

Esta técnica permite transformar una secuencia obtenida por cualquier método y que sigue la distribución uniforme, en otra que siga otro tipo de distribución. Si pretendemos generar un número aleatorio que cumpla con una determinada función acumulada de probabilidad F(x) podemos proceder de la siguiente forma (representada además en la figura 2.1):

Paso 1	Generar un número d que siga la distribución uniforme, empleando para ello cualquiera de las técnicas vistas al principio de este capítulo (método del medio del cuadrado, congruencial, etc).
Paso 2	A partir de d, obtener un número x_0 siguendo la distribución no uniforme F deseada mediante $x_0 = F^{-1}(d)$.

Es decir, para generar una secuencia de números aleatorios que cumplan una determinada función de probabilidad acumulada F(x), generamos una secuencia de números aleatorios que sigan la distribución uniforme con cualquiera de los métodos ya conocidos, y los hacemos pasar a través de la función $F^{-1}(x)$, obteneniendo así otra secuencia con distribución F(x). Este método es muy útil cuando disponemos de la función de distribución acumulada en forma tabulada. En este caso, la función inversa se obtiene muy fácilmente invirtiendo la abcisa y la ordenada de cada uno de los puntos de la distribución tabulada.

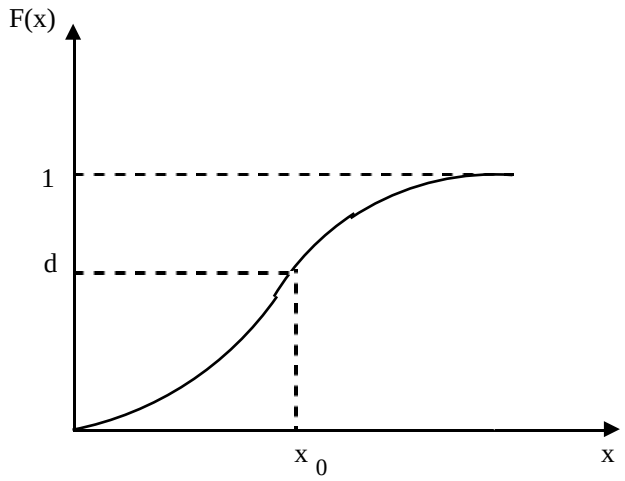


Fig. 2.1 Función acumulada de probabilidad

EjJEMPLO 2.8: Supongamos que disponemos de la distribución tabulada en la tabla 2.1 y queremos obtener otro número que cumpla esta distribución. Para ello, y mediante cualquiera de los métodos vistos en este capítulo, generamos un número que siga la distribución uniforme. Si el número obtenido con distribución uniforme es, por ejemplo, 0,25, el correspondiente número de la distribución F(x) se obtendría interpolando entre los valores 1,0 y 1,5 de F(x), y obtendríamos de esta forma el número 1,0833.

Tabla 2.1

x	0,0	0,5	1,0	1,5	2,0	2,5	3,0
F(x)	0,0	0,1	0,2	0,5	0,7	0,9	1,0

2.2.2 Obtención de números según la distribución uniforme no estándar

La función de distribución uniforme no estándar viene determinada por la expresión:

$$F(x)=\begin{cases} 0 & x<a \\ (x-a)(b-a) & a\le x\le b \\ 1 & x>b \end{cases}$$

Para generar una secuencia de números aleatorios que sigan esta distribución, partiendo de otra secuencia obtenida por otros métodos que cumpla con la distribución uniforme estándar, basta con escalar los números de dicha secuencia, de forma que si d es un número de dicha distribución uniforme estándar, $x = a+(b-a)d$ será un número aleatorio que siga la distribución uniforme no estándar en el rango a...b.

2.2.3 Obtención de números según la distribución gausiana (normal)

Existen diversos métodos para obtener secuencias de números pseudoaleatorios con distribución normal a partir de otras con distribución uniforme estándar. Uno de ellos consiste en utilizar la transformación inversa descrita antes. Sin embargo, no existe una forma cerrada de representar la función de distribución acumulada normal, por lo que no es posible deteminar $F^{-1}(x)$, y ello obliga a utilizar una representación de dicha distribución normal tabulada que incluya tantos más puntos cuanto mayor sea el grado de detalle que se desee obtener. La tabla 2.2 muestra una tabulación de dicha distribución.

EJEMPLO 2.9: Supongamos que hemos generado mediante el método congruencial lineal los números aleatorios con distribución uniforme estándar 0,2163, 0,3241, 0,1021 y 0,7621, y queremos obtener, a partir de ellos, una secuencia de números que sigan una distribución normal de media (m) 3 y desviación típica (σ) 4. Empleando la tabla 2.2, los números de la distribución normal correspondientes a los obtenidos son (interpolando) -0,8077, -0,4594, -1,3070 y 0,7357.

Ya se puede convertir esta distribución normal de media 0 y varianza 1 a la deseada mediante la relación $y = m + \sigma x$, con $m = 3$ y $\sigma = 4$, obteniendo la secuencia 1,3846, 2,0812, 0,3842 y 4,4714, que ya cumple la distribución deseada.

Otro método muy utilizado para transformar secuencias con distribución uniforme estándar en otras con distribución normal es la que usa el teorema central del límite, que establece que la suma de n variables aleatorias independientes e idénticamente distribuídas $X_1, X_2,..., X_n$ se comporta aproximadamente como una distribución normal con media nm y varianza $n\sigma^2$, donde m y σ^2 son las medias y las varianzas de X_i, $i = 1...n$. Si $X_1, X_2,..., X_n$ siguen además la distribución uniforme estándar, entonces $m = 1/2$ y $\sigma^2 = 1/12$.

Hay que tener además en cuenta que cuanto mayor sea el número n de variables que se sumen, tanto más exacta será la aproximación. Un valor interesante puede ser $n = 12$, ya que mantiene un buen compromiso entre el grado de aproximación y una buena eficiencia de cálculo.

El método para obtener una secuencia de números con distribución normal de media m y varianza σ^2 será, pues, el siguiente:

Paso 1	Generar una secuencia de números pseudoaleatorios $\{X\}$ con distribución uniforme estándar empleando algún método como el medio del cuadrado, métodos congruenciales, etc.
Paso 2	Agrupar los números de la secuencia $\{X\}$ en grupos de n números.
Paso 3	Sumar todos los números pertenecientes a cada grupo y obtener la secuencia $\{Y\}$, en la que los elementos son las sumas obtenidas. Observar que la secuencia $\{Y\}$ tendrá distribución normal de media n (1/2) y varianza n (1/12).
Paso 4	Transformar la secuencia $\{Y\}$ en otra $\{Z\}$ que tenga distribución normal con media 0 y varianza 1 aplicando a cada número de la secuencia la expresión $Z_i = Y_i - n(1/2)$.
Paso 5	Obtener finalmente la secuencia $\{S\}$ con la distribución normal de media m y varianza σ^2 deseada aplicando a cada número de la secuencia $\{Z\}$ la expresión $S_i = m + \sigma Z_i$.

EJEMPLO 2.10: Supongamos que obtenemos la secuencia de números 0,1062, 0,1124, 0,7642, 0,4314, 0,6241, 0,9443, 0,8121, 0,2419, 0,3124, 0,5412, 0,6212 y 0,0021, que presenta una distribución normal estándar, mediante algún método conocido y queremos, a partir de ella, generar un número que siga la distribución normal de $m = 25$ y $\sigma^2 = 9$.

Puesto que disponemos de $n = 12$ números, se realiza una suma sobre ellos con lo que se obtiene el número $Y = 5,5135$, que pertenecerá a una distribución normal de media 12(1/2) = 6 y varianza 12(1/12) = 1. Se efectúa la operación $Z = Y-6 = -0,4865$, con lo que se obtiene un número de la distribución normal de media 0 y varianza 1.

Finalmente, aplicando a este número la expresión $S = m + \sigma Z$ con los valores de $m = 25$ y $\sigma = 9$, se obtiene el número 23,5405, que ya pertenece a la distribución normal de media 25 y varianza 9 que deseábamos.

Tabla 2.2 Distribución normal tabulada

x	*F(x)*
-3	0,00135
-2.5	0,00621
-2.0	0,02275
-1.5	0,06681
-1.0	0,15866
-0.5	0,30854
-0.0	0,50000
0.5	0,69146
1.0	0,84134
1.5	0,93319
2.0	0,97725
2.5	0,99379
3.0	0,99865

2.2.4 Obtención de números según la distribución binomial

Esta distribución se emplea para modelar un experimento consistente en *n* sucesos sucesivos, que pueden cada uno de ellos dar dos posibles resultados. Una variable aleatoria que contará el número de veces que se produce un determinado suceso con probabilidad *p* en cada uno de los *n* intentos que se realizan, estaría distribuída según una distribución binomial. Este hecho se puede aprovechar para generar secuencias de números simulando el experimento anterior y suponiendo que los *n* resultados se obtienen empleando un generador de números aleatorios distribuídos uniformemente. Una vez obtenidos estos *n* valores uniformemente distribuídos, se pueden obtener números distribuídos binomialmente simplemente contando cuántos de los *n* números obtenidos son menores o iguales que *p*. Es decir, para obtener una secuencia de números distribuídos binomialmente, con probabilidad *p*, se puede realizar:

Paso 1 Generar una secuencia $\{X\}$ de números pseudoaleatorios uniformemente distribuídos empleando, por ejemplo, algún método como el medio del cuadrado o cualquier método congruencial.

Paso 2 Agrupar los números de la secuencia $\{X\}$ en grupos de *n* bits, teniendo en cuenta que cada grupo producirá un número de una secuencia $\{Y\}$ distribuída binomialmente con probabilidad *p*.

Paso 3 Obtener cada número de la secuencia $\{Y\}$ contando, dentro de cada grupo generado a partir de la secuencia $\{X\}$, cuántos números son menores que *p*.

Este procedimiento será válido cuando el valor de *n* que se elija para realizar los grupos que simulan cada experimento sea pequeño (por ejemplo <10). Sin embargo, si se quieren obtener mejores aproximaciones se deberán elegir valores de *n* más grandes. Entonces, es más conveniente emplear la aproximación normal que la distribución binomial que, en vez de generar números distribuídos binomialmente, lo que hace es generar números que sigan una distribución nomal con media *np* y varianza *np*(1-*p*), empleando para ello el algoritmo del apartado anterior.

EJEMPLO 2.11: Supongamos que se ha generado la secuencia de números 0,02011, 0,85393, 0,97265, 0,61680, 0,16656, 0,42751 y 0,69994, distribuídos uniformemente a partir del método congruencial lineal. Si queremos obtener a partir de ellos un número que siga una distribución binomial con $p = 3$ a partir de estos $n = 7$ números, basta con mirar cuántos de ellos son menores que $p = 0,3$. Puesto que sólo 2 lo son, éste será el nuevo número obtenido ($Y = 2$).

2.2.5 Obtención de números según la distribución exponencial

La función de probabilidad acumulada exponencial para una variable aleatoria *X* presenta la forma $F(x) = 1 - e^{\alpha x}$, para $x > 0$. Para obtener secuencias de números que sigan esta distribución lo más sencillo es aplicar el método de la transformación inversa. Para ello se necesita dicha función inversa que es $F^{-1}(z) = -ln(z)/\alpha$. Puesto que partiremos de una secuencia $\{Z\}$ de números pseudoaleatorios uniformemente distribuídos, una secuencia constituída por los números $1\text{-}z_i$ también tendría esta distribución y, por tanto, se puede expresar $F^{-1}(z)$ como $F^{-1}(z) = -ln(z)/\alpha$. El algoritmo para generar una secuencia de números pseudoaleatorios con distribución exponencial con parámetro α será:

Paso 1 Generar una secuencia $\{Z\}$ de números pseudoaleatorios con distribución uniforme mediante algún metodo conocido (por ejemplo algún método congruencial).

Paso 2 Obtener la secuencia de números pseudoaleatorios $\{X\}$ con distribución exponencial de parámetro α aplicando a los números de la secuencia $\{Z\}$ la transformación:

$$x_i = -\frac{1}{\alpha} 1n(z_i)$$

Este algoritmo es muy fácil de programar, pero tiene el inconveniente de que consume mucho tiempo de ejecución, debido a que debe calcular el logaritmo natural. Si esto se convirtiera en un problema, se puede recurrir a la utilizción de una distribución exponencial en forma tabulada, y emplear entonces el método de la transformación inversa descrito en el apartado 2.2.1.

EJEMPLO 2.12: Si tenemos el número $z = 0,02104$ obtenido a partir del método congruencial lineal y, por tanto, perteneciente a una distribución uniforme, a partir de él podemos obtener un número *z* que pertenezca a una distribución exponencial con parámetro $\alpha = 1$, haciendo $x = -ln(0,02104) = 3,8613$.

2.2.6 Obtención de números según la distribución de Poisson

La generación de una secuencia de números pseudoaleatorios que sigan una distribución de Poisson con media λ, se puede realizar multiplicando sucesivos números que sigan una distribución de tipo uniforme estándar según el procedimiento siguiente:

Paso 1	Generar una secuencia $\{X\}$ de números pseudoaleatorios que sigan una distribución uniforme estándar mediante algún método conocido (por ejemplo cualquier método congruencial).
Paso 2	Agrupar los números de la secuencia $\{X\}$ en grupos de M números (R_1, R_2,.., R_M).
Paso 3	Para un grupo, realizar la multiplicación de N de sus números consecutivos ($N < M$) mientras se cumpla la condición: $$\prod_{i=1}^{N} R_i < e^{-\lambda}$$
Paso 4	Generar un número de la secuencia $\{Y\}$ que tenga una distribución de Poisson con media λ, asignando $Y_i = N$-1, donde N se obtuvo en el paso anterior.
Paso 5	Seleccionar el siguiente grupo de M bits de la secuencia $\{X\}$ y volver al Paso 3 para obtener el siguiente número de la secuencia $\{Y\}$.

EJEMPLO 2.13: Mediante el método congruencial lineal se ha generado la secuencia 0,91646, 0,89198, 0,64809, 0,16376, 0,91782, 0,45624 y 0,31641, que presenta una distribución uniforme. A partir de estos números queremos obtener otro que siga una distribución de Poisson con $\lambda = 2{,}5$. Puesto que $e^{-2,5} = 0{,}08208$, se multiplicarán los diferentes números mientras el resultado sea menor que 0,08208:

(N = 2) 0,91646 x 0,89198 = 0,81746 > 0,08208
(N = 3) 0,91646 x 0,89198 x 0,64809 = 0,52979 > 0,08208
(N = 4) 0,91646 x 0,89198 x 0,64809 x 0,16376 = 0,08675 > 0,08208
(N = 5) 0,91646 x 0,89198 x 0,64809 x 0,16376 x 0,91782 = 0,7963 < 0,08208

Puesto que la condición se cumple para N = 5, podemos generar un número Y que pertenezca a una distribución de Poisson de media 2,5 haciendo $Y = N$-1 = 4.

2.3 Problemas

PROBLEMA 2.1 Sea un generador de secuencias pseudoaleatorias lineal mixto de la forma $x_{n+1} = (ax_n+b) \bmod m$, $n \geq 0$.a) Obtener los 10 primeros números aleatorios que producirá el generador en el

caso que $a = 2$, $b = 3$, $m = 10$ y la semilla $x_0 = 0$. b) Realizar lo mismo que en el caso a) pero con los valores $a = 2$, $b = 0$, $m = 10$ y $x_0 = 1$ (notar que se trata de un generador congruencial multiplicativo).

PROBLEMA 2.2 ¿.Cuál es la longitud del periodo de una secuencia obtenida mediante un generador de congruencias lineales si $a = 3141592621$, $b = 2718281829$, $m = 10000000000$ y $x_0 = 5772156648$?

PROBLEMA 2.3 ¿Son las siguientes condiciones suficientes para garantizar un periodo de longitud máxima en un generador congruencial mixto cuando $m = 2^e$ es una potencia de 2?: (1) b es impar. (2) $a \bmod 4 = 1$.

PROBLEMA 2.4 En la práctica, la generación de números pseudoaleatorios mediante algoritmos congruenciales lineales mixtos se realiza usando la expresión $x_{n+1} = (ax_n+b) \bmod m$, donde las x son enteros, y luego se normalizan al intervalo de reales entre 0 y 1 mediante la división $u_n = x_n/m$. Discutir la generación de estos números reales normalizados al intervalo [0,1) si ésta se realiza directamente mediante la recursión $u_{n+1} = (au_n+b/m) \bmod 1$ usando la aritmética en coma flotante del ordenador.

PROBLEMA 2.5 Discutir la mejora en la velocidad de generación de los números pseudoaleatorios utilizando el algoritmo de MacLaren-Marsaglia si las secuencias $\{X_n\}$ e $\{Y_n\}$ son la misma.

PROBLEMA 2.6 Obtener una secuencia de 15 números pseudoaleatorios empleando el método del medio del cuadrado si la semilla es $x_0 = 0{,}3245$.

PROBLEMA 2.7 Generar una secuencia de 10 dígitos pseudoaleatorios empleando el método congruencial lineal con semilla $x_0 = 21$, $a = 4$, $c = 1$ y $m = 100$.

PROBLEMA 2.8 Emplear el método de la transformación inversa para obtener una secuencia de números pseudoaleatorios que cumplan con la función de densidad de probabilidad acumulada:

$$F(x) = \begin{cases} 1/4 & 0 \le x < 1 \\ 3/4 & 1 \le x \le 2 \end{cases}$$

PROBLEMA 2.9 A partir de una secuencia de números pseudoaleatorios con distribución uniforme obtenidos por cualquier método: (a) generar un número pseudoaleatorio con distribución normal de media 15 y varianza 5; (b) generar un número pseudoaleatorio con distribución binomial con $n = 8$ y $p = 0{,}28$;(c) generar un número pseudoaleatorio con distribución exponencial con $\alpha = 2{,}3$; (d) generar un número pseudoaleatorio con distribución de Poisson con media $\lambda = 6$.

2.4 Bibliografía

[AKLME] AKL, S. G.; MEIJER, H. *A Fast Pseudo Random Permutation Generator with Applications to Cryptology*. Dept. of Comput. and Inform. Science. Queen´s University. Kingston, Ontario, Canada

[COV60] COVEYOU, R. R. *Serial Correlation in the Generation of Pseudo-Random Numbers*. J. ACM 7, pp. 72-74, 1960.

[FEL68] FELDER, H. *The GPSS/360 Random Number Generator*. Digest of the Second Conference of Applications of Simulation, New York, Dec. 1968.

[FISHM] FISHMAN, G. S.; MOORE, L. R. *A Statistical Evaluation of Multiplicative Congruential Random Number Generators with Modulus* 2^{31}*-1*.

[FOR91] FORRE, R. The *Hénon Attractor as a Keystream Generator*. Eurocrypt `91. Brghton, April, 1991.

[GOR61] GORDON, G. *A General Purpose Systems Simulation Program*. Proc. EJCC, Washington, D.C. New York. MacMillan, 1961.

[GOR71] GORDON, G. *A General Purpose Systems Simulation Program*. Proc. EJCC, Washington, D.C. New York: Macmillan,1971.

[HEN76] HENON, M. *A two-dimensional Mapping with a Strange Attractor*. Communications in Mathematical Physics, pp. 59-77, 1976.

[HUL62] HULL, T. T.; DOBELL, A. R. *Random Number Generators*. SIAM Rev. 4 , pp. 230-254, July 1962.

[HUT66] HUTCHINSON, D. W. *A New Uniform Pseudo Random Number Generator*. Commun. ACM 9, 6, pp. 432-433, June 1966.

[KNO70] KNOTT, G. D. *A Numbering System for Permutations and Combinations*. Communications of the ACM, Vol. 19, No. 6, June 1976.

[KNU67] KNUTH, D. E. *The Art of Computer Programming. Vol 1: Seminumerical Algorithms.* Adison-Wesley 1967.

[KNU81] KNUTH, D. E. *The Art of Computer Programming. Volume 2: Seminumerical Algorithms.* Addison-Wesley, 1981

[LEC88] L'ECUYER, P. *Efficient and Potable Combined Random Number Generators.* Commun. ACM 31, 6, Pgs. 742-749, 774, June 1988.

[LEH51] LEHMER, D. H. *Proceedings of the Second Sysmposium on Large-Scale Digital Computing Machinery*. Cambridge, Mass: Harvard University Press, 1951.

[LEW69] LEWIS, P. A.; GOODMAN, A. S. *A Pseudo-Random Number Generator for the System/360*. IBM Syst. J. 8, pp. 136-146, Feb. 1969.

[MAC65] MACLAREN, M. D.; MARSAGLIA, G. *Uniform Random Number Generators*. J. ACM 12, 1. Jan.1965. Pags. 83-89.

[MAR68] MARSAGLIA, G. *One-Line Random Number Generators and Their Use in Combinations.* Commun. ACM 11, 11, pp. 757-759. Nov.1968.

[PAY69] PAYNE, W.H.; RABUG, J. R.; BOGYO, T. P. *Coding the Lehmer Pseudo Random Number Generator*. Commun. ACM 12, 2, pp. 85-86, Feb. 1969.

[PRA88] PRAK, S. T.; MILLER, K. W. *Random Numbers Generators: Good Ones are Hard to Find.* Communications of the ACM. October 1988, Volume 31, Number 10. SIAM J. Sci. Stat. Comput. 7, 1 (1986) Pgs. 24-25.

[ROM90] ROMERA, M.; JIMENEZ, I. NEGRILLO, J. *Generación de Secuencias Cifrantes Mediante Funciones Caóticas*. I Reunión Española de Criptología. Mallorca, 1990.

[SCH79] SCHRAGE, L. *A More Portable FORTRAN Random Number Generator*. ACM Trans. Math. Sotfw. 5, 2, pp. 132-138.June 1979.

[WIC82] WICHMANN, B. A.; HILL, I. D. *An Efficient and Portable Pseudo Random Number Generator.* Appl. Stat. 31. pp. 188-190.1982.

3 Técnicas de diseño de generadores

3.0 Introducción

En el capítulo anterior se estudiaron diversos algoritmos generadores de secuencias pseudo-aleatorias propuestos de forma específica para ser ejecutados sobre un ordenador. Sin embargo, en la mayoría de aplicaciones de comunicaciones, tanto comerciales como militares, que hacen uso de este tipo de secuencias, se emplean métodos *hardware* para generarlas, con lo que se consigue no sólo eficiencia en cuanto a la velocidad de generación de los números aleatorios, sino también simplicidad y bajo coste.

Existen gran cantidad de métodos para generar secuencias pseudoaleatorias, la mayoría de ellos basados en un dispositivo denominado registro de desplazamiento con realimentación lineal (y que denotaremos como LFSR de su nombre en inglés). Tal como se verá, con este dispositivo se pueden generar secuencias de números con una aleatoriedad lo suficientemente buena como para satisfacer los requerimientos de la mayoría de aplicaciones de comunicaciones que necesitan de este tipo de secuencias.

Sin embargo, a pesar de las buenas propiedades estadísticas de las secuencias obtenidas mediante estos dispositivos, éstas presentan una elevada predictibilidad, es decir, que conocida una cierta parte de la secuencia generada por el LFSR, es fácil obtener el resto de la secuencia. En ciertas aplicaciones, como el cifrado en flujo, (que se estudiará con más detalle en el capítulo 7) que hace uso de este tipo de secuencias para proteger los datos que se deben transmitir, el hecho de que las secuencias producidas por los LFSR sean altamente predictibles será un problema grave que habrá que solucionar, de forma que la seguridad del sistema sea realmente efectiva. En los últimos años se han desarrollado diversas técnicas, como por ejemplo la combinación no lineal del contenido de las celdas de los LFSR, la mezcla mediante funciones combinadoras apropiadas de las salidas de diversos LFSRs o el uso de técnicas de control del reloj en estos dispositivos. El objetivo de todas estas técnicas, que se van a detallar en este capítulo, es aumentar la impredictibilidad de las secuencias pseudoaleatorias producidas, pero sin estropear las buenas propiedades estadísticas de los LFSRs que se utilizan como elementos básicos.

3.1 Registros de desplazamiento con realimentación lineal

Los LFSR [BAR58][GOL67] están constituídos por un conjunto de L etapas o celdas de memoria interconectadas mediante puertas lógicas OR-exclusivas (EX-OR). Las celdas de memoria están unidas entre sí, de tal forma que cada pulso de reloj recibido hace avanzar el contenido de cada una de ellas y la carga en la celda adyacente correspondiente, en función del sentido de avance elegido. La primera celda

se carga con el valor obtenido del resultado de la operación EX-OR de realimentación, con lo que queda el sistema con un camino cerrado por esta realimentación, tal como se ve en la figura 3.1.

La justificación para el uso de este tipo de dispositivos está en que son modelables por una máquina de estados finitos con un número limitado de estados y de entradas, de forma que, dados un estado y entrada determinados, el nuevo estado de la máquina quedará completamente determinado. A partir de esto, es posible construir una tabla con todos los estados posibles o bien un diagrama de estados que muestre, para cada estado posible, a qué nuevo estado se pasará cuando llegue cada una de las posibles entradas.

Los parámetros que caracterizarán un determinado LFSR serán su longitud (o número de celdas de memoria) y las celdas que intervendrán en la función lineal de realimentación .

Veamos la teoría que nos permite describir este tipo de dispositivos. Se considera un LFSR de L etapas (L se denomina también el *orden* del LFSR), y se llama a_i a los coeficientes de realimentación, $a_i \in \{0,1\}$ (1 si ha conexión y 0 si no la hay), y sea r_i el contenido de las celdas de memoria del registro de desplazamiento $r_i \in \{0,1\}$, tal como muestra la figura 3.1, $r_{i(t)}$ es el contenido de la celda r_i después del pulso t-ésimo. Se puede representar la función de realimentación como $f(t) = a_0 r_0(t) \oplus a_1 r_1(t) \oplus \ldots \oplus a_{L-1} r_{L-1}$. Si se asume que $a_0 = 1$, el nuevo bit de salida del LFSR $r_{L-1}(t+1)$ dependerá siempre de $r_0(t)$. Con $a_i = 1$ se denota una conexión cerrada y con $a_i = 0$ se denota una conexión abierta.

Un registro de desplazamiento de L celdas produce una secuencia de vectores del espacio $(r_0(t), r_1(t), \ldots, r_{L-1}(t))$ (con t = 0, 1, 2,...), y cumplirá las siguientes relaciones:

1) $r_i(t+1) = r_{i+1}(t) \qquad i = 0,1,\ldots,L-2$
2) $r_{L-1}(t+1) = \sum_{i=0}^{L-1} a_i r_i(t) \qquad t = 0, 1, 2, \ldots$
3) $z(t) = r_0(t) \qquad t = 0,1,2\ldots.$
4) $z(t)$ tiene periodo $\leq 2^L - 1$.

donde $z(t)$ es la secuencia de salida. Se pueden identificar las conexiones de realimentación con los coeficientes de un polinomio f(x) en el cual, si existe un término determinado, consideramos que la celda asociada a él contribuye a la obtención de la realimentación. A este polinomio se le suele denominar polinomio característico. El registro de desplazamiento se puede caracterizar mediante este polinomio, que se expresa como $f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{L-1} x^{L-1} + x^L$, con $a_0 = 1$. El hecho de que $a_0 = 1$ garantizará que se aproveche toda la longitud del LFSR a la hora de hacer la realimentación, mientras que el término x^L se corresponde con la conexión de realimentación. Por tanto, $r_{L-1}(t+1)$ siempre dependerá de $r_0(t)$.

El conjunto de todas las secuencias finitas binarias generadas mediante el polinomio f(x) se llama espacio de soluciones y se denota por Ω(f), que es un espacio n-dimensional sobre los campos de Galois GF(2) en el que se definen las operaciones multiplicación ($\cdot$) y suma módulo 2 ($\oplus$).

Una cuestión importante que se debe tener en cuenta es que la salida del registro de desplazamiento es periódica, lo que quiere decir que cada celda de memoria repetirá su contenido cada cierto tiempo. Este hecho nos aparta del caso ideal de no periodicidad para las secuencias verdaderamente

aleatorias que vimos en el capítulo 1, ya que se limita el número de estados diferentes del LFSR y que dependerá, además, de cuál sea su número de celdas (su orden), de su estado inicial y de las conexiones del polinomio de realimentación.

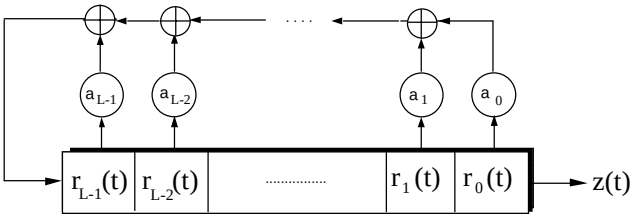


Fig. 3.1 Registro de desplazamiento con realimentación lineal (LFSR)

Para un registro de desplazamiento con realimentación lineal, si f(*x*) tiene grado *L* y es primitivo, el periodo de cualquier secuencia no nula será $P = 2^L-1$, es decir, tendrá la máxima longitud. Un *polinomio primitivo* de grado *L* será un polinomio irreducible sobre el cuerpo finito GF(2) que, cuando se utilice para producir la realimentación de un registro de desplazamiento de *L* celdas, haga que la salida de éste tenga periodo máximo 2^L-1.

Esto implica que el registro de desplazamiento recorrerá 2^L-1 estados no nulos antes de repetirse. El hecho de que sean 2^L-1 estados y no 2^L se debe a que el estado inicial todo ceros debe evitarse siempre, ya que llevaría al LFSR a generar continuadamente la secuencia cero.

A las secuencias obtenidas por LFSR con polinomio de realimentación primitivo se las llama *m-secuencias* o también secuencias G-aleatorias, debido a que cumplen a la perfección los tres postulados de Golomb. En la elección del polinomio característico, además de ser primitivo para tener máximo periodo, deberemos intentar que tenga el mínimo número posible de conexiones para que el dispositivo obtenido sea lo más rápido y económico posible.

Tal como se ha visto antes, las operaciones que se pueden realizar con los polinomios que se emplean para representar las funciones de realimentación de los LFSR se definen sobre campos de Galois [BEN76]. A continuación se describirán este tipo de estructuras.

3.1.1 Bases matemáticas de los cuerpos finitos

En este apartado se definirán, de forma simple, algunas características básicas sobre las estructuras llamadas cuerpos finitos y que normalmente se denotan por GF(q). Dentro de estas estructuras, son de especial interés las de la forma $GF(2^n)$, debido a que su implementación electrónica se puede realizar fácilmente utilizando elementos biestables. Los cuerpos finitos encuentran gran aplicabilidad en campos como la conmutación, la codificación, la criptografía, etc.

Los cálculos que se pueden hacer sobre un campo finito (o campo de Galois) GF(q) son, básicamente, la suma, el producto, el simétrico y el inverso. La suma de dos elementos de GF(q) es la suma coordenada a coordenada (en paralelo) de elementos de GF(q), aunque normalmente, y éste es

nuestro caso, nos centraremos en los campos de orden 2, es decir, GF(2). Esto implica trabajar en el caso binario, donde los únicos elementos que consideramos son el 1 y el 0. De esta forma, la suma anterior se consigue fácilmente mediante una puerta electrónica XOR (OR-Exclusiva). El producto y el inverso son más difíciles de conseguir en GF(q), aunque si nos restringimos al caso de campos GF(2), que sólo trabajan con dos elementos, el producto se podrá realizar empleando una puerta AND, y el inverso con una puerta inversora. Este tipo de estructuras simples permitirán trabajar con más comodidad a la hora de tratar matemáticamente los dispositivos que se usan para generar secuencias pseudoaleatorias. Veamos la teoría básica de los cuerpos finitos.

Si q es un número primo, definimos F al conjunto $F=\{0, 1,..., q-1\}$ de enteros menores que él. Se puede definir el sistema algebraico $<F, \oplus , \cdot >$ en el cual se definen, sobre dos elementos a y b de F ($a,b \in F$) las siguientes operaciones:

$$a \oplus b = R_q (a + b)$$

$$a \cdot b = R_q (ab)$$

donde $R_q(z)$ es el resto de la división del entero z por el primo q. Se denota entonces por GF(q) al campo de Galois de orden q (campo finito con q elementos) [LEM71].

El caso más importante para nosotros será el campo GF(2), en el cual el conjunto F se reduce a dos elementos, cero y uno. En este campo, las reglas de suma y multiplicación se basan en la aritmérica módulo 2. Por tanto, en GF(2), cero y uno serán los dígitos binarios.

En el caso general GF(q), un polinomio con variable x se puede expresar como: $A(x) = a_0 + a_1x + ... + a_nx^n$, donde n es un entero no negativo, y los coeficientes a_i para $0 \le i \le n$ serán elementos de GF(q), a_n será el coeficiente de mayor grado, y a_0 será el coeficiente constante.

El polinomio cuyos coeficientes son todo ceros es el polinomio nulo y su grado será $-\infty$ por convenio. Además, los polinomios de grados menores o iguales que cero se llaman polinomios constantes. También sobre los polinomios sobre GF(q) se pueden definir las operaciones suma y producto. Si $A(x)$ y $B(x)$ son dos polinomios de grados n y m respectivamente (con $n \ge m$) sobre GF(q), se define la suma polinómica como:

$$A(x) + B(x) = \sum_{i=0}^{n} (a_i \oplus b_i) \cdot x^i$$

que consiste en la suma, módulo 2, de los coeficientes del mismo orden de cada uno de los polinomios, siendo a_i y b_i los coeficientes de los polinomios $A(x)$ y $B(x)$ respectivamente.

Análogamente, se puede definir la multiplicación como:

$$A(x)B(x) = \sum_{k=0}^{n+m} c_k x^k \qquad \text{si} \qquad c_k = \sum_{\substack{i+j=k \\ 0 \le i \le n \\ 0 \le j \le m}} a_i \cdot b_j$$

Entonces, el conjunto de todos los polinomios de grado finito sobre GF(q) con las operaciones suma y multiplicación polinómicas forma un anillo polinómico sobre $GF_{(q)}(x)$ [DAIZD].

3.1.2 Polinomio característico primitivo

Se ha visto que para que la secuencia producida por un LFSR sea de longitud máxima, es decir, 2^L-1 si el LFSR tiene L celdas, el polinomio de realimentación debía ser primitivo. Veamos en qué consiste la primitividad de polinomios, pero para ello se definirá en primer lugar qué es un polinomio irreducible.

Se definen los polinomios irreducibles sobre un campo como los polinomios que no pueden ser separados en factores de grado positivo sobre dicho campo, es decir, si no es posible factorizarlos. La irreductibilidad de polinomios es, pues, una propiedad análoga a la primitividad de los enteros.

Ya se ha visto en el apartado anterior que no todas las conexiones de realimentación de un LFSR llevan a secuencias de salida de periodo máximo. Para la realización práctica de la mayoría de sistemas, interesará que la longitud de la secuencia del LFSR sea la máxima posible, circunstancia que se dará tan sólo cuando el polinomio de realimentación, además de ser irreducible, sea primitivo. Veamos la definición de polinomio primitivo:

> Si un polinomio irreducible f(x) = x^m+.... divide a x^v+1 con $v = 2^m$-1, y no divide a x^v+1 con $v < 2^m$-1, entonces se puede generar todo el cuerpo a partir de él, y se le llama *polinomio primitivo*.

Así pues, una vez comprobado que un polinomio de grado L sobre GF(2) es irreducible, deberemos comprobar además que la secuencia de salida del LFSR es de longitud máxima para garantizar que el polinomio es primitivo [SUR78]. En la práctica se recurre directamente a polinomios primitivos que se pueden encontrar en tablas.

Es de gran interés determinar el número de polinomios de realimentación primitivos de los que se puede disponer para cada orden L. Para ello, se puede usar la función de Euler que establece para un entero positivo z, cuántos enteros positivos menores que él $\phi(z)$ son mutuamente primos con z. Por tanto, la función de Euler $\phi(L)$ nos permitirá conocer el número $\lambda(L)$ de polinomios de realimentación primitivos que podremos encontrar para un cierto orden L (es decir, para un LFSR de L celdas, de forma que éste pueda producir secuencias de salida de periodo máximo 2^L-1):

$$\lambda(L) = \varnothing\left(2^L - 1\right) / L$$

A un número primo de la forma 2^L-1 se le llama un primo de Mersenne y se cumple que si 2^L-1 es primo, entonces L también debe serlo. Esta propiedad no se cumple, sin embargo, en el sentido contrario (es decir, el hecho de que b sea un número primo no implica que 2^b-1 también deba serlo). Sólo hay 27 primos de Mersenne con $L < 45000$. La tabla 3.1 muestra el número de polinomios primitivos que se pueden obtener para órdenes entre 3 y 11.

De cara a obtener cierta comodidad en el diseño de secuencias pseudoaleatorias producidas por LFSRs, al final del capítulo se incluye una tabla (tabla 3.3) con polinomios primitivos para grados entre 2 y 17. Si bien esta tabla no es exhaustiva, se pueden encontrar tablas más completas de polinomios primitivos para órdenes de hasta 55 [HER82].

Tabla 3.1 Número de polinomios primitivos para diversos órdenes

Nº de celdas (L)	*Longitud de secuencia (2^L-1)*	*Nº Polinomios primitivos λ (L)*
3	7	2
4	15	2
5	31	6
6	63	6
7	127	18
8	255	16
9	511	48
10	1023	60
11	2047	176

3.1.3 Propiedades de las *m*-secuencias

Las secuencias obtenidas mediante LFSR con polinomios primitivos presentan una serie de propiedades que las caracterizan y las hacen particularmente interesantes:

1) El periodo de las *m*-secuencias obtenidas por un LFSR de L celdas es de 2^L-1.

2) En cada periodo de la *m*-secuencia hay 2^{L-1} unos y 2^{L-1}-1 ceros.

3) En cada periodo hay 2^{L-k} copias de cada estructura de k bits, excepto para la de k ceros, de la cual hay 2^{L-k}-1 copias, con $k \leq L$.

Estas propiedades, junto con su facilidad de obtención hacen que los LFSR sean muy interesantes para obtener secuencias pseudoaleatorias. Para ver si una secuencia es semejante a la que obtendríamos al lanzar monedas al aire debemos comprobar si cada bit de nuestra secuencia tiene información dependiente de otros bits de la secuencia. Una medida interesante de este tipo de dependencia es la que nos da la función de autocorrelación de la secuencia. En el caso de *m*-secuencias de periodo $P = 2^L$-1, la autocorrelación fuera de fase es constante y de valor -1/P, tal como establecía el tercer postulado de Golomb. Por tanto, las *m*-secuencias satisfacen bastante bien las propiedades de aleatoriedad, pero siempre hasta cierto grado debido a su longitud finita, por lo que la autocorrelación (y por tanto la dependencia entre bits) no llega a ser cero.

3.1.4 Impredictibilidad de las secuencias y complejidad lineal (Λ)

Debido a la linealidad que presenta la generación del nuevo bit que servirá de entrada al LFSR, la salida de éstos es altamente predecible. Esta predictibilidad de la secuencia será un factor crítico en ciertas

aplicaciones como la criptografía, y habrá que buscar técnicas que permitan disminuírla. Una medida que permitirá determinar el grado de impredictibilidad de las secuencias pseudoaleatorias es la llamada *complejidad (*o *equivalente) lineal* Λ [GOL89], que se estudiará más profundamente en el capítulo 6. Para que la secuencia producida por un LFSR sea impredictible, deberá presentar una complejidad lineal elevada, aunque esto será tan sólo una condición necesaria y no suficiente. Además, será necesario que, independientemente del bit precedente, el bit siguiente de la secuencia aparezca con una distribución uniforme. Una razón más para usar la complejidad lineal como parámetro de medida de la impredictibilidad de la secuencia es que el periodo de la secuencia de salida del LFSR sea al menos tan grande como la complejidad lineal, por lo que una gran complejidad lineal implicará siempre un periodo grande. Las secuencias pseudoaleatorias muestran un aumento de la complejidad lineal al aumentar el número de dígitos de la secuencia. Esto no ocurre siempre a la inversa, pues una secuencia periódica no implica necesariamente gran complejidad lineal.

Las secuencias de máxima longitud o *m*-secuencias obtenidas a partir de LFSRs con polinomio de realimentación primitivo presentan excelentes propiedades respecto a la distribución estadística, pero tienen una complejidad lineal mínima respecto a la longitud del periodo que vale:

$$\Lambda = L$$

donde L es el número de celdas del LFSR. Esto implica que, conociendo tan sólo $2L$ bits de la secuencia de salida, empleando el algoritmo de síntesis de Massey-Berlekamp [MAS69] se pueden obtener el estado inicial y las conexiones de realimentación del LFSR que la generó, y puede obtenerse el resto del periodo de la secuencia.

Para secuencias aleatorias, podemos esperar además que el crecimiento de dicha complejidad siga un perfil irregular, pero siguiendo la línea de $n/2$, donde n es el número de bits de la secuencia. Por tanto, si la secuencia es periódica, para tener una buena impredictibilidad, la complejidad debe crecer según esta línea y alcanzar el valor del periodo una vez se hayan procesado 2 periodos de la secuencia utilizando el algoritmo anterior.

Está claro que cuanto mayor es el periodo, o cuanto mayor es la complejidad lineal de la secuencia, mayor será su impredictibilidad. Sin embargo con los LFSR siempre se estará limitado en cuanto a la complejidad ($\Lambda \leq L$). Si se desean obtener secuencias con mayor impredictibilidad para aplicaciones como la criptografía deberemos recurrir a técnicas no lineales que, aplicadas sobre las salidas de los LFSR produzcan mayores complejidades ($\Lambda > L$).

La complejidad o equivalente lineal seguirá siendo una medida muy interesante de impredictibilidad para las secuencias obtenidas de forma no lineal. En este caso, dicho equivalente será simplemente el LFSR de menor orden capaz de generar dicha secuencia obtenida mediante técnicas no lineales.

Tal como se verá en el capítulo 7, en aplicaciones criptográficas como el cifrado en flujo, el hecho de que las secuencias pseudoaleatorias utilizadas presenten elevada complejidad será un requisito imprescindible, hasta el punto que si no se alcanzan valores elevados se estropeará completamente la efectividad del sistema, y se convertirá en muy vulnerable. En este caso sería muy fácil para posibles atacantes violar su seguridad. La importancia de esta propiedad en este tipo de sistemas ha impulsado la aparición de gran número de técnicas no lineales que, aplicadas sobre los LFSR, aumentan la complejidad de las secuencias de salida. De forma genérica, las tres técnicas más importantes son la aplicación de funciones no lineales sobre las celdas de los LFSRs, la combinación no lineal de las salidas de varios

LFSRs y la aplicación de un control no lineal en los relojes que atacan a los LFSRs. Todas estas técnicas se van a ver a continuación, mientras que en el capítulo siguiente se verán muchos generadores de secuencias pseudoaleatorias basados en ellas. La dificultad en la aplicación de todas estas técnicas no lineales radica en conseguir aumentar la complejidad de las secuencias de salida de los LFSR aumentando así su impredictibilidad, pero sin estropear sus buenas propiedades estadísticas. Esto hará del diseño de estos generadores una tarea delicada.

3.2 La función de estado no lineal f

Veamos en este apartado las funciones no lineales f aplicadas sobre las celdas de un único LFSR con motivo de aumentar la impredictibilidad de la secuencia producida por éste [GRO71][KEY76][YOE61]. La dificultad estriba en conseguir funciones no lineales que no estropeen las buenas propiedades estadísticas que presentan los LFSR.

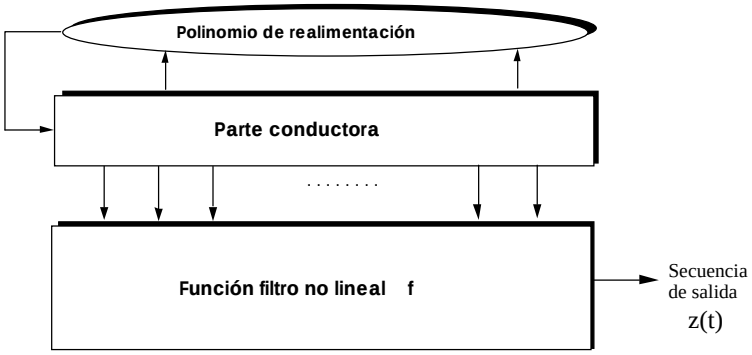


Fig. 3.2 La función de estado no lineal

Para estudiar los generadores de secuencias pseudoaleatorias del tipo no lineal, es conveniente subdividirlos en una parte conductora y en una parte combinadora de las diferentes celdas del registro de desplazamiento, tal como muestra la figura 3.2. La parte conductora se encarga de gobernar la secuencia de estados del generador y es la responsable de conseguir el periodo máximo de la secuencia. La parte combinadora de las diferentes fases del registro de desplazamiento tiene como función principal la tarea de incrementar la complejidad lineal de la secuencia de salida, pero sin modificar las buenas propiedades de distribución estadística de ceros y unos obtenidas mediante el uso de conexiones de realimentación por polinomios primitivos de la parte conductora.

La única memoria del sistema está incluída por las celdas del registro de desplazamiento, puesto que tanto el polinomio primitivo de realimentación, como la función de estado no lineal, se aplican sobre dichas celdas (a las que se suelen llamar fases debido a la propiedad de desplazamiento del LFSR), y actúan de manera directa en función del estado interno que tenga el sistema en cada momento, tal como se ve en la figura 3.2. Por tanto, las funciones de estado no lineal deben utilizar, tal como indica su nombre, una transformación no lineal sobre las celdas de los LFSR que debe cumplir las siguientes condiciones:

1) Debe transferir las propiedades estadísticas de las secuencias periódicas obtenidas de la parte conductora en el sentido de que, si las secuencias de entrada son *k*-distribuídas (tal como se definió en el capítulo 1), las secuencias de salida también deben serlo.

2) El periodo de la secuencia de salida obtenido mediante la parte conductora (LFSR con polinomio de conexión primitivo) no debe ser alterado por la función no lineal.

3) Debe maximizar la complejidad lineal respecto a la de la secuencia obtenida por la parte conductora.

4) Deben ser rápidas en su funcionamiento para ser aplicables a sistemas en tiempo real y fáciles de implementar.

5) Su realización debe ser lo más económica posible, ya que en muchas aplicaciones éste es uno de los factores determinantes de todo el sistema.

Lo que se hace al introducir funciones de estado no lineales en GF(2) es aumentar la dificultad en el análisis de la secuencia de salida, pero no aumentar el periodo de la misma (de hecho, a veces disminuye). Se consigue simular secuencias que sólo pueden ser generadas por LFSR de órdenes muy superiores al realmente implementado. Al añadir a un LFSR la lógica binaria adicional no lineal, se consigue aumentar la complejidad lineal o equivalente lineal del mismo, pero manteniendo las propiedades de distribución estadística de ceros y unos, así como su número absoluto, dentro de un periodo de la secuencia, y la longitud máxima de la secuencia.

Un ejemplo simple de función no lineal podría ser el producto del contenido de dos celdas del LFSR de orden *L* con polinomio primitivo. Esto haría que la complejidad de la secuencia de salida fuese:

$$\Lambda = \binom{L}{1} + \binom{L}{2}$$

lo que implica que necesitaríamos un LFSR de dicho orden para poder generar la misma secuencia que la obtenida mediante la introducción de la no linealidad. Si añadiésemos más celdas del LFSR al producto anterior, se añadirían nuevos términos binomiales a la suma del equivalente anterior, es decir:

$$\left(\binom{L}{3}, \binom{L}{4}, \ldots\right)$$

Dicho proceso se podría extender hasta que todos los elementos no cero de GF(2^L) estuvieran presentes, momento en el cual no podría elevarse más el equivalente lineal. Las operaciones no lineales más simples en GF(2) serán el producto de dos dígitos binarios ($f(x_1, x_2) = x_1 \cdot x_2$, donde $\cdot$ representa la función AND) y la suma de dos dígitos binarios ($f(x_1, x_2) = x_1 \oplus x_2$, donde $\oplus$ representa la suma XOR).

La función no lineal se puede expresar en una forma canónica llamada la Forma Algebraica Normal (ANF) , que refleja directamente la no linealidad en forma de sumas de productos, es decir, expresa cualquier función no lineal en función de las dos operaciones básicas disponibles en GF(2): $\cdot$ y $\oplus$. Veamos con un ejemplo cómo poner las funciones no lineales en forma de sumas de productos (notar que con suma nos referimos a la suma módulo 2, es decir, la operación OR-exclusiva).

EJEMPLO 3.1: Dado el siguiente generador de secuencias pseudoaleatorias, poner la función no lineal en la forma algebraica normal (ANF).

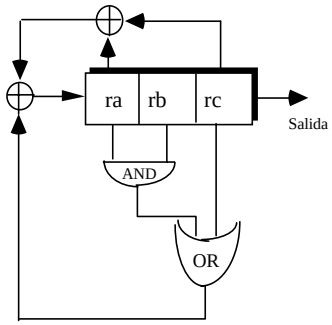


Fig. 3.3

Puesto que en los campos de Galois GF(2), sólo están definidas las operaciones producto y suma binaria módulo 2 ($\oplus$), hemos de poner la operación suma OR (+) en función de estas operaciones. Puesto que $x+y = x\cdot y \oplus x \oplus y$, la función no lineal queda como: $r_a \cdot r_b \cdot r_c \oplus r_a \cdot r_b \oplus r_c$, que se puede poner como:

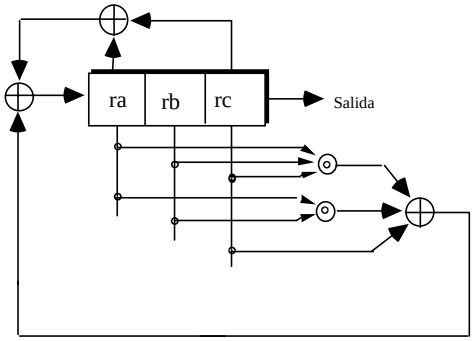


Fig. 3.4

El generador que se obtiene usando las no linealidades se llama generador pseudoaleatorio si no existe ningún algoritmo probabilístico en tiempo polinomial que, para cualquier longitud deseada, pueda distinguir entre la secuencia obtenida y la secuencia realmente aleatoria.

El orden de la función no lineal f por sí solo no nos va a poder garantizar el valor máximo de la complejidad lineal de la secuencia producida. Sin embargo, lo que sí podemos saber con certeza es que el producto de orden *L* de todas las celdas del LFSR producirá una secuencia con complejidad lineal máxima e igual al periodo. Sin embargo, hay que tener cuidado ya que, aunque en una primera aproximación pueda parecer que ya se ha obtenido el objetivo de conseguir la complejidad lineal máxima, y por tanto la máxima impredictibilidad, la secuencia resultante estará formada por 2^{L-1}-1 ceros y tan sólo un uno, por lo que habrá perdido completamente las buenas propiedades estadísticas del LFSR. Esto nos lleva a que las transformaciones que usen tan sólo el producto de orden máximo son peligrosas y deben ser evitadas. En la práctica, resultará interesante emplear transformaciones no lineales que incorporen entre otros a varios productos de fases, próximos al orden máximo. Las propiedades que muestra la función no lineal f dependerá fuertemente del polinomio particular del LFSR generador. Por tanto, a la hora de estudiar un generador de estas características, habrá que considerar las diferentes relaciones estructurales entre los polinomios de conexión y las funciones no lineales consideradas. Esto nos lleva a que será muy difícil realizar un tratamiento determinista de las secuencias obtenidas mediante un filtrado no lineal, a causa de las posibles degeneraciones que puedan aparecer, y tendremos que conformarnos con realizar un estudio de tipo probabilista. Con dicho estudio probabilista, lo que se pretenderá es hacer arbitrariamente pequeña la probabilidad de que aparezcan estas degeneraciones. Para ello, podemos definir algunas cotas a la complejidad [DAI90]. Para una función no lineal f de orden *k* (producto de *k* fases de un LFSR de *L* celdas), la complejidad lineal de la secuencia producida estará acotada superiormente por la expresión:

$$\Lambda \leq \sum_{i=1}^{k} \binom{L}{i}$$

Sin embargo, para aplicaciones prácticas lo que verdaderamente resultará interesante será poder determinar una cota inferior a dicha complejidad. Desgraciadamente, dicha complejidad no está acotada inferiormente debido a que pueden aparecer degeneraciones que nos aparten de los valores máximos deseables para la misma. Existen, no obstante, casos particulares para los que sí es posible establecer unas cotas inferiores para la complejidad lineal, como son los casos de productos de fases equidistantes y la combinación lineal de productos.

3.2.1 Producto de fases equidistantes

En este caso, la no linealidad aplicada a las fases del LFSR, consiste en tomar los contenidos de celdas equidistantes del mismo y realizar entre ellas la operación producto lógico binario. En estas condiciones, la condición necesaria para conseguir una complejidad lineal:

$$\Lambda \geq \binom{L}{k}$$

(donde *k* es el número de fases tomadas para realizar la operación producto), es que la distancia entre cada par de fases que contribuyan a la obtención de la salida no lineal estén separadas una distancia δ que sea prima relativa respecto al valor del periodo 2^{L}-1.

3.2.2 Combinación lineal de productos

Este caso sería el mismo que el anterior, pero añadido un nuevo término, resultante de la aplicación de la operación producto sobre un número N de celdas consecutivas del LFSR. En este caso, la complejidad lineal estará acotada inferiormente por

$$\Lambda \geq \binom{L}{k} - (N-1)$$

Mientras el número de fases N sea menor que la distancia k entre fases sucesivas, se podrá mantener el límite inferior de la complejidad. En caso de no ser así, se deberá disminuir algo este límite.

3.3 Combinación no lineal de varios LFSR

La finalidad de una función combinadora no lineal C es combinar las secuencias de salida de N generadores de secuencias pseudoaleatorias (figura 3.5), y su objetivo es el mismo que se buscaba con el empleo de funciones no lineales f. El hecho de utilizar funciones combinadoras C permitirá ampliar el orden del campo de Galois sobre el que se trabaja.

Básicamente, la función combinadora C combinará un cierto número N de secuencias binarias de entrada que generalmente estarán producidas por LFSRs o LFSRs con L_i $(1 \leq i \leq N)$ celdas respectivamente y con filtro de estado no lineal f. En este caso la complejidad lineal de la secuencia resultante de aplicar la operación combinadora C estará completamente determinada por las funciónes no lineales f y por las complejidades lineales de todas las secuencias concurrentes. Esto será así si, además, se cumple que los polinomios característicos de conexión de los registros de desplazamiento son primitivos y tienen longitudes u órdenes primos relativos entre sí. Una función combinadora C debe cumplir una serie de propiedades que se presentan a continuación:

1) La secuencia de salida debe mostrar elevado periodo. El máximo posible será: $P_{max} = 2^{\sum_{i=1}^{N} L_i - 1}$

2) La secuencia de salida debe tener una elevada complejidad lineal Λ. La máxima complejidad lineal que se podrá alcanzar será $\Lambda_{\max} \approx P_{\max}$.

3) Los unos y los ceros en la secuencia de salida deberán ser igualmente probables.

4) La estructura obtenida debe mostrar la mayor inmunidad a la correlación posible.

Si bien el concepto de inmunidad a la correlación se describirá en el siguiente apartado, se dice que una función combinadora C presenta inmunidad a la correlación de orden m cuando cualquier conjunto de m entradas $\{x_1(t), x_2(t),..., x_m(t)\}$ a la función combinadora están incorreladas con su salida $z(t)$.

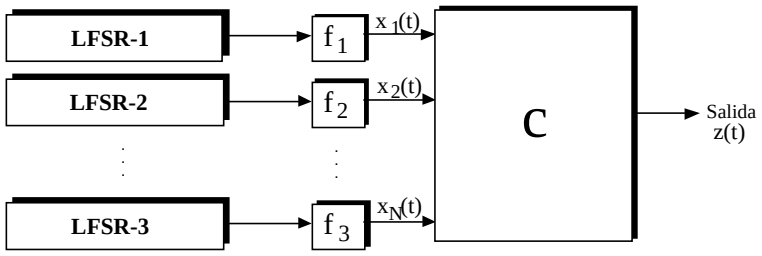


Fig. 3.5 Función combinadora de varios LFSRs con función no lineal

Esta propiedad será de gran importancia cuando se diseñen generadores basados en una función *C* para aplicaciones criptográficas, ya que el hecho de que existan dependencias entre las entradas a *C* y su salida puede ser aprovechado por un posible atacante (que hubiese podido capturar dicha salida o una porción de ella) para obtener información de la estructura completa del generador.

3.3.1 La inmunidad a la correlación

La inmunidad a la correlación es una propiedad importante de las funciones combinadoras *C*, que tendrá gran importancia cuando se empleen dichas funciones para realizar generadores de secuencias pseudoaleatorias para aplicaciones como la criptografía. Veamos este concepto. Para ello, se considera que las secuencias de entrada a la función *C* son independientes y están distribuídas uniformemente.

Se dice entonces que una función combinadora no lineal *C* sin memoria presenta inmunidad a la correlación de orden *m* [SIE84] si la salida de dicha función combinadora y *m* cualesquiera de las secuencias que le sirven de entrada, consideradas en su conjunto, son estadísticamente independientes. Es decir, si la información mutua (I) entre la salida y cualquier subconjunto de *m* secuencias de entrada es cero:

$$I\left(z_i : x_{1_i}, x_{2_i}, \ldots, x_{m_i}\right) = 0$$

Sin embargo, el hecho de conseguir que se cumpla esta propiedad tendrá un precio, que vendrá dado por la limitación que aparecerá en el máximo orden de la no linealidad de *C* que se podrá alcanzar. Siegenthaler [SIE84] demuestra que existe un compromiso entre el orden *k* alcanzable por la no linealidad de la función combinadora *C* de *N* entradas y el grado de inmunidad a la correlación alcanzable *m*. Este compromiso viene dado por la siguiente relación:

$$k + m \leq N$$

Si además se requiere que la secuencia de salida *z*(t) de la función combinada *C* esté uniformemente distribuída, la relación de compromiso anterior deberá ser:

$$k + m \leq N \qquad \text{para} \quad m = 0 \quad \text{ó} \quad m = N - 1$$
$$k + m \leq N - 1 \qquad \text{para} \quad 1 \leq m \leq N - 2$$

La máxima inmunidad a la correlación que se podrá alcanzar será de orden N - 1 y corresponderá a la suma (módulo 2) de todas las variables de entrada a la función combinadora C.

El hecho de tener que reducir el orden de la no linealidad en el caso de que se requiera aumentar la inmunidad a la correlación implicará necesariamente disminuir la complejidad lineal de la secuencia de salida de la función combinadora. Por ejemplo, si se desea obtener inmunidad a la correlación mínima (de orden 1) y salida de la función combinadora C uniformemente distribuída, se deberá reducir el orden de la no linealidad de C hasta al menos N - 2.

Por lo tanto, el compromiso entre el orden de la no linealidad y la inmunidad a la correlación de C se podrá ver también como un compromiso entre dicha inmunidad a la correlación y la complejidad lineal de la secuencia de salida de C.

Este compromiso, que aparece en funciones combinadoras C sin memoria, desaparecerá si se elimina dicha restricción. Si se añade cierta memoria a la función combinadora C (y para ello bastará con un solo bit de memoria), será posible obtener al mismo tiempo máxima complejidad lineal de la secuencia de salida e inmunidad a la correlación máxima de orden N - 1.

3.3.2 Funciones combinadoras más utilizadas

Si se combinan N registros de desplazamiento con realimentación lineal sobre GF(q), todos ellos con polinomio de realimentación primitivo y de longitudes diferentes entre sí y mayores que dos mediante una función combinadora C no lineal, se alcanzará una gran complejidad lineal en la secuencia de salida.

Las funciones más simples que se pueden emplear como funciones combinadoras son el producto y la suma binarias, sin embargo, estas funciones combinadoras presentan problemas en cuanto a la distribución de ceros y unos en las secuencias de salida, y será interesante estudiar otras funciones más complejas para implementar dicha función como las básculas, los multiplexores, los codificadores o los contadores cíclicos.

A) Suma y producto de secuencias pseudoaleatorias

Estas dos operaciones constituyen la forma más simple de combinar las secuencias producidas por varios registros de desplazamiento de realimentación lineal (LFSR). La principal ventaja de emplear estas funciones combinadoras es su sencillez, tanto de funcionamiento como de implementación, usando circuitería electrónica. Veamos en primer lugar las propiedades que presenta el producto de dos secuencias producidas por LFSR´s, cuyo esquema se puede ver en la figura 3.6. Si los periodos de los registros de desplazamiento con realimentación lineal son T_1 y T_2 respectivamente, el periodo T de la secuencia de salida del dispositivo estará acotado por la siguiente expresión:

$$\frac{T_1 T_2}{mcd\left(T_1, T_2\right)} < T < T_1 T_2$$

Si además, las secuencias producidas por los dos registros de desplamiento con m y n celdas respectivamente son m-secuencias, es decir, los polinomios de realimentación que emplean son primitivos, el periodo y la complejidad lineal de la secuencia de salida del dispositivo de la figura 3.6 serán:

$$T = \left(2^m - 1\right)\left(2^n - 1\right)$$
$$\Lambda = m \cdot n$$

donde $T_1 = 2^m-1$ y $T_2 = 2^n-1$, si m y n son el número de celdas de ambos registros de desplazamiento respectivamente y se cumple que T_1 y T_2 son mutuamente primos entre sí. Esta última condición se puede expresar como mcd(2^m-1, 2^n-1)=1, lo que también implica que mcd(m,n)=1, donde mcd es la abreviatura del máximo común divisor.

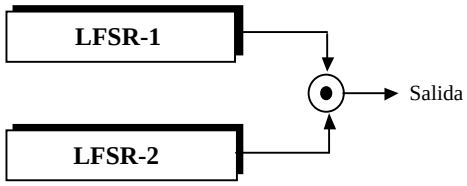


Fig. 3.6 Producto de secuencias pseudoaleatorias

Sin embargo, el problema que presenta el empleo de la operación producto como función combinadora de las secuencias producidas por registros de desplazamiento con realimentación lineal, es que la secuencia de salida presenta muy mala distribución estadística de ceros y unos, debido a la propia naturaleza del producto de dos números binarios (3/4 de las salidas serán 0 ya que $0\times 0 = 0$, $0\times 1 = 0$, $1\times 0 = 0$ y $1\times 1 = 1$). Veamos ahora las propiedades de la suma binaria (OR-exclusiva) como función combinadora de secuencias producidas por registros de desplazamiento, tal como muestra la figura 3.7.

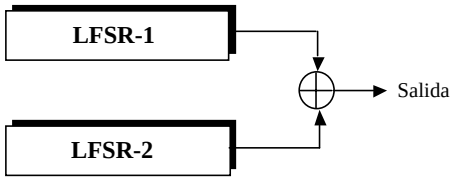


Fig. 3.7 Suma (módulo 2) de secuencias pseudoaleatorias

Si los periodos de ambos registros de desplazamiento son T_1 y T_2 y son mutuamente primos entre sí (es decir, mcd(T_1T_2=1), entonces el periodo de salida será $T = T_1T_2$. En general, para N secuencias producidas por LFSR´s con periodos mútuamente primos entre sí, se cumple que:

$$T = \prod_{i=1}^{N} T_i$$

y la complejidad lineal de la secuencia de salida viene acotada según la expresión:

$$\Lambda \leq \left(2^m - 1\right)\left(2^n - 1\right)$$

donde *m* y *n* son el número de celdas de ambos registros de desplazamiento respectivamente.

La suma módulo 2 de secuencias binarias no presenta el problema de desbalanceo estadístico entre los unos y los ceros que presentaba la operación de multiplicación que se ha visto anteriormente, puesto que en este caso las combinaciones que dan cero y uno a la salida están equilibradas ($0\times 0 = 0$, $0\times 1 = 1$, $1\times 0 = 1$, $1\times 1 = 0$).

B) El multiplexor como función combinadora

Los multiplexores son dispositivos electrónicos que obtienen una única salida a partir de un conjunto de entradas, tal como muestra la figura 3.8.

Mediante el valor que se den a unas líneas de control, se seleccionará una u otra línea de entrada cuyo contenido será depositado en la línea de salida. Se pueden aprovechar las propiedadas no lineales de este dispositivo para combinar de forma adecuada diferentes secuencias pseudoaleatorias de entrada en cada una de las líneas del multiplexor.

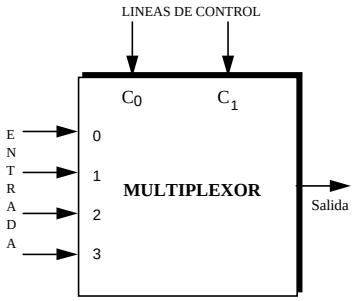


TABLA DE ESTADOS DEL MULTIPLEXOR

CONTROL C0	CONTROL C1	Entrada conectada a la salida
0	0	0
0	1	1
1	0	2
1	1	3

Fig. 3.8 Multiplexor como función combinadora

Con *N* líneas de control se puede seleccionar una entre 2^N posibles líneas de entrada para asignar su valor a la salida. Como es lógico, se buscará que todo el proceso se parezca lo más posible a uno que genere secuencias verdaderamente aleatorias, por lo que se intentará producir procesos aleatorios allí donde sea posible.

Firmes candidatos a esto, son tanto las líneas de control como las líneas de entrada del multiplexor. Podemos pensar en diferentes estructuras para generar secuencias pseudoaleatorias:

ESTRUCTURA A) Líneas de entrada con valores fijos y líneas de control gobernadas mediante las celdas de un LFSR, o bien celdas de distintos LFSRs, tal como muestra la figura 3.9.

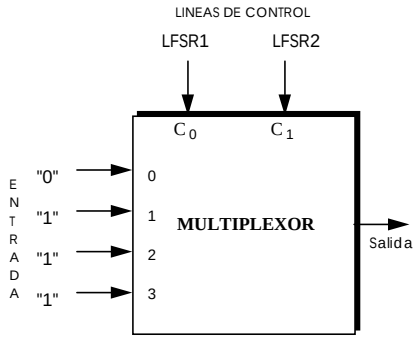


Fig. 3.9 Estructura A

ESTRUCTURA B) Líneas de control gobernadas por un contador que cambie de estado progresiva y cíclicamente, y líneas de entrada atacadas por una serie de LFSRs (figura 3.10).

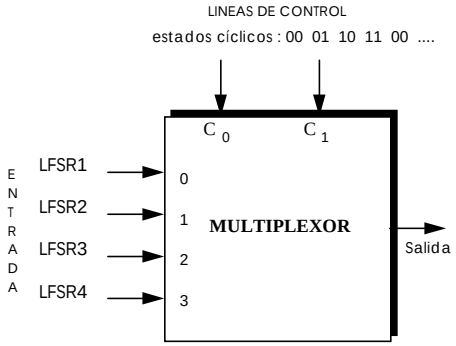


Fig. 3.10 Estructura B

ESTRUCTURA C) Líneas de control y de entrada atacadas por las celdas de distintos registros de desplazamiento (figura 3.11).

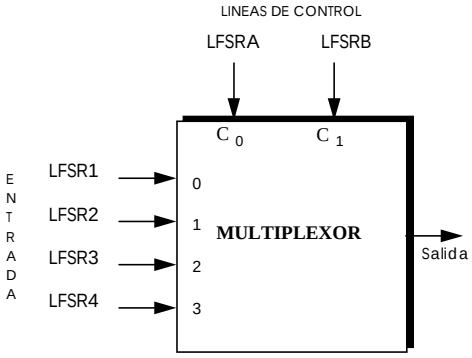


Fig. 3.11 Estructura C

ESTRUCTURA D) Líneas de control gobernadas por un LFSR y líneas de entrada por otro (figura 3.12).

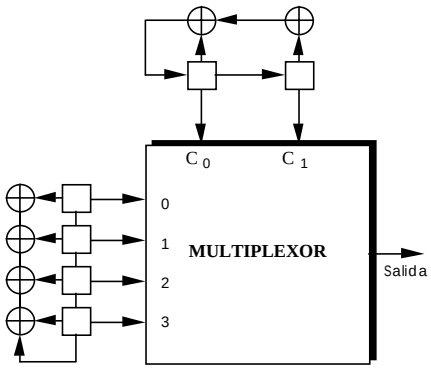


Fig. 3.12 Estructura D

Vamos a ver algunos resultados en cuanto a periodo y complejidad lineal sobre la estructura D. Para ello, consideremos dos LFSR que tienen L_1 y L_2 celdas respectivamente, de forma que las celdas del $LFSR_1$ ataquen a las líneas de entrada del multiplexor. Si el multiplexor tiene N líneas de control, dispondremos de 2^N líneas de entrada, de las cuales se podrá seleccionar una para que se conecte a la salida mediante las líneas de control. Si llamamos $LFSR_2$ al registro de desplazamiento que genera las señales que controlarán las líneas de selección, debemos usar N celdas de las L_2 que posee dicho registro. Si para las líneas de entrada empleamos las celdas del $LFSR_1$, se tendrán que utilizar 2^N celdas de las L_1 de que consta para completar las líneas de entrada, el contenido de una de las cuales se pasará a la salida en cada instante de avance del reloj del sistema.

Con éstas, ya se tienen las primeras restricciones en cuanto a los órdenes de los LFSR que se deben emplear: $L_2 \geq N$ y $L_1 \geq 2^N$.

Además, para obtener las condiciones óptimas en cuanto a periodo y complejidad lineal será interesante que los LFSRs tengan polinomio de realimentación primitivo. Con esta configuración, la longitud de la secuencia de salida del multiplexor, será periódica y su periodo estará acotado por la expresión:

$$P \leq (2^{L_1}-1)(2^{L_2}-1)$$

Cuando los periodos de las secuencias producidas por los LFSRs que sirven de entrada, o lo que es lo mismo, los órdenes de ambos LFSRs, son mútuamente primos entre sí (mcd(L_1,L_2) =1), la secuencia de salida del sistema presentará el máximo periodo:

$$P = P_1 \cdot P_2 \quad \text{si} \quad P_1 = 2^{L_1}-1 \quad \text{y} \quad P_2 = 2^{L_2}-1$$

Hay que tener en cuenta al diseñar una estructura de este tipo que si L_1 y L_2 son divisibles, el periodo de la secuencia de salida se verá muy reducido:

$$P \leq \text{mcm}((2^{L_1}\text{-}1)\ ,\ (2^{L_2}\text{-}1))$$

Es interesante notar que en esta estructura, el periodo P de la secuencia de salida no depende del polinomio característico primitivo elegido para los registros de desplazamiento ni del número de entradas de control N que tenga el multiplexor.

Otro parámetro de interés es el comportamiento en cuanto a la autocorrelación que presentará la secuencia de salida. Para esta estructura, la autocorrelación para valores fuera de fase valdrá aproximadamente:

$$\frac{1}{P_1^2} - \frac{1}{P_1 P_2}$$

que representa un valor muy pequeño, lo que nos sugiere que la secuencia de salida transporta poca información de los polinomios primitivos utilizados. Además, cuando se cumpla la restricción $L_1 > L_2$, la correlación valdrá aproximadamente:

$$\frac{-1}{\left(2^{L_2} - 1\right)}$$

En cuanto a la complejidad lineal λ de la secuencia de salida de este sistema con multiplexor, se tendrá:

a) $\lambda = L_2\left(1 + L_1\right)$ si $N = 1$

b) $\lambda = L_2\left(1 + L_1 + \left(\frac{L_1}{1}\right)\right)$ si $N = 2 \leq L_1$

c) $\lambda \leq L_2\left(1 + \sum_{i=1}^{N}\left(\frac{L_1}{i}\right)\right)$ si $2 < N < L_1\text{-}1$

La ecuación c) se cumplirá con igualdad si las N celdas de las que se extrae su contenido para atacar las líneas de entradas del multiplexor se toman de forma equiespaciada.

d) $\lambda = L_2\left(2^{L_1} - 1\right)$ si $N = L_1\text{-}1$ ó $N = L_1$

C) Las básculas como funciones combinadoras

Otro ejemplo de funciones combinadoras lo representan las que usan básculas binarias biestables *J-K* para combinar las secuencias obtenidas por varios registros de desplazamiento con realimentación lineal. Puesto que, a diferencia de las funciones combinadoras vistas hasta ahora, este tipo de dispositivos tienen memoria interna, se podrá conseguir simultáneamente máxima complejidad lineal e inmunidad total a la correlación. La tabla 3.2 muestra las transiciones que se producirán en la salida de la báscula según cuál sea el valor que tomen las dos entradas J y K.

Tabla 3.2 Transiciones de una báscula JK

ENTRADAS		*ESTADO*
J	*K*	*SIGUIENTE*
0	0	q_{n-1}
0	1	0
1	0	1
1	1	$\overline{q_{n-1}}$

Por tanto, las salidas de este dispositivo no sólo dependen de la entrada, sino también del estado anterior. Además, presenta la desventaja de que hay que fijar un estado inicial q_{n-1}.

Veamos cómo utilizar este dispositivo para combinar diversas secuencias pseudoaleatorias. La figura 3.13 muestra una estructura en la que se emplean dos registros de desplazamiento ($LFSR_1$ y $LFSR_2$) para atacar las entradas *J* y *K* de la báscula.

Si el número de celdas de ambos registros de desplazamiento con realimentación lineal L_1 y L_2 son mutuamente primos entre sí, es decir $mcd(L_1,L_2) = 1$, entonces el periodo que se obtendrá con esta estructura será:

$$P_{\max} = \left(2^{L_1} - 1\right)\left(2^{L_2} - 1\right)$$

Además, el uso de básculas *J-K* como funciones combinadoras hace que el generador de secuencias pseudoaleatorias obtenido presente las siguientes características:

1) El tiempo de ejecución será elevado (aunque este problema sería salvable mediante el empleo de tecnologías más avanzadas).
2) Presenta un estado inicial transitorio (un preámbulo), aunque este efecto será sólo apreciable al arrancar el sistema y durará unos pocos bits.
3) Toda la secuencia de salida depende del estado inicial que le demos al sistema.
4) No se alcanza una distribución uniforme a la salida a no ser que las secuencias concurrentes estén bastante balanceadas. (En el caso de la figura 3.13 se conseguirá ya que las secuencias concurrentes son LFSRs que presentan una buena distribución en sus secuencias de salida).

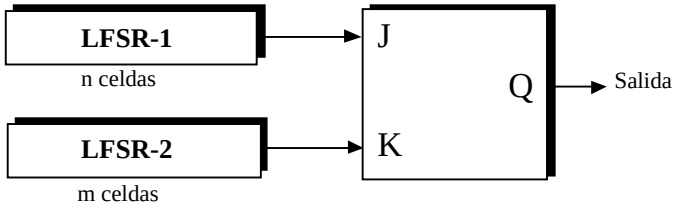


Fig. 3.13 Báscula JK como función combinadora

D) Funciones combinadoras con multiplicadores y sumadores

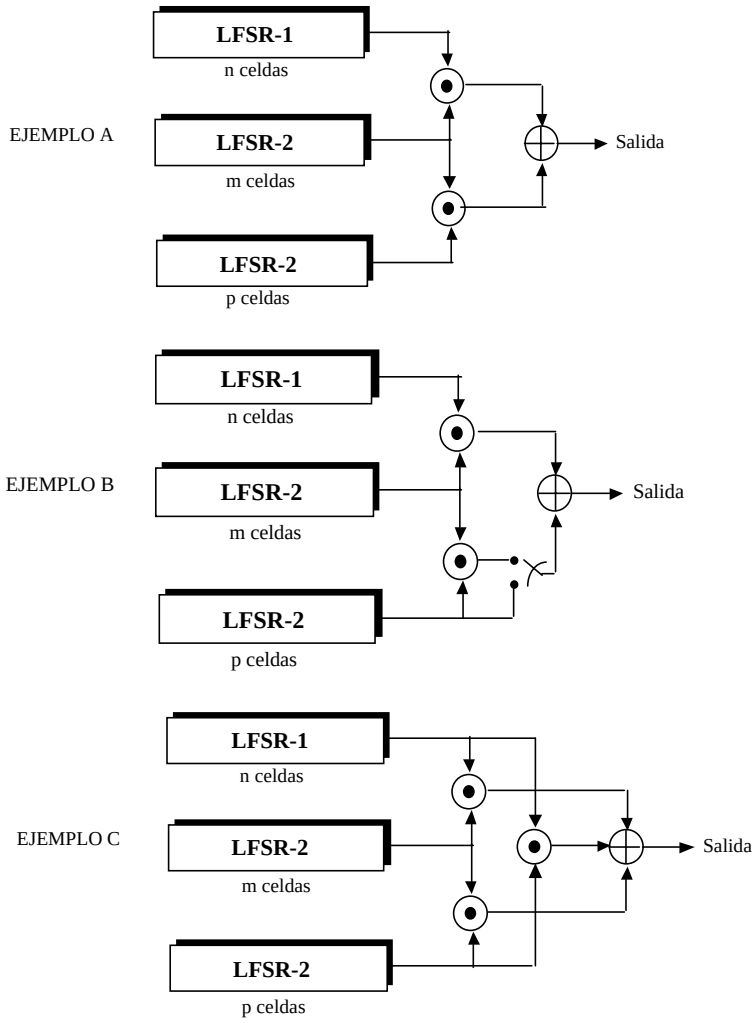


Fig. 3.14 Diversos casos de multiplicadores y sumadores como funciones combinadoras

El uso de funciones combinadoras que emplean sumadores y multiplicadores simultáneamente evita (mediante compensación) la mala distribución de ceros y unos que presentaban las secuencias de salida vistas en el apartado 3.3.1, al emplear estas operaciones separadamente. Las diversas conexiones que se puedan sugerir vendrán limitadas por la idea de que ninguno de los registros de desplazamiento con realimentación lineal que usemos debe dominar sobre los otros en la secuencia de salida. A este método de generación de funciones combinadoras mediante la asociación de sumas y productos se le suele llamar mapeo simétrico de los LFSR. La figura 3.14 muestra algunas posibles estructuras de estas características.

3.4 Técnicas de control de reloj

Otra forma de aumentar la complejidad lineal de las secuencias de salida de los registros de desplazamiento consiste en la manipulación del reloj que controla el cambio de estado de éstos [CHA88][GOL89]. Con el uso de este tipo de técnicas no sólo se aumenta la complejidad, sino que se pueden simular nuevas secuencias de salida. La mayor parte de estos dispositivos basan su funcionamiento en introducir algún sistema que convierta en no lineal el reloj principal. La no linealidad consistirá básicamente en intentar conseguir un reloj pseudoaleatorio.

Físicamente, podemos realizar el reloj controlador mediante algún tipo especial de contador, pero en general se utilizarán LFSRs en cascada, de forma que la salida de cada uno alimente la entrada de reloj del siguiente. Este esquema en cascada será el habitual en generadores que empleen este tipo de técnicas. Las variaciones diferenciadoras entre diversas opciones serán cuestiones tales como la longitud de los LFSR elegidos, su número o cuántos avances del LFSR principal correspondan para cada estado de salida del LFSR de control. Los tipos de control de reloj se pueden dividir en tres tipos diferentes que se verán en los apartados siguientes.

3.4.1 Controladores de marcha y espera (*stop & go*)

En este tipo de generadores [BET85], el reloj que hace avanzar al LFSR o al sistema de registros de desplazamiento que hayamos implementado es controlado por otro dispositivo, que en general estará constituído por otro LFSR, tal como muestra la figura 3.15. De este modo, el sistema generador de la secuencia avanzará su estado interno actual al estado siguiente solamente cuando el LFSR controlador genere un bit uno en su secuencia de salida, que es la entrada de reloj del sistema de registros. En el capítulo siguiente se estudiarán diversas implementaciones de generadores basados en esta técnica.

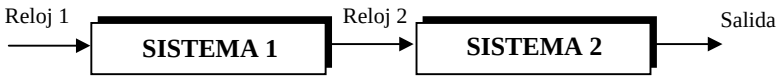


Fig. 3.15 Controlador de marcha y espera

3.4.2 Control de reloj en cascada

En esta clase de dispositivos [GOL85] controladores de reloj no se efectúa una acción directa sobre el reloj del LFSR o del sistema de registros de desplazamiento, sino que, mediante otro dispositivo formado por registros de desplazamiento, al cual se le hace funcionar con el mismo reloj que el sistema generador, actuaremos de alguna forma (A) sobre la secuencia de salida de este último, tal como se muestra en la figura 3.16.

Con este método se consigue que absolutamente todos los registros de desplazamiento que empleamos realicen los cambios de sus estados internos correspondientes con la misma periodicidad, dada por el reloj principal del sistema.

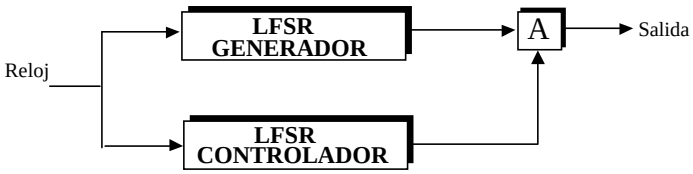


Fig. 3.16 Control de reloj en cascada

3.4.3 Variador de la velocidad del reloj

Con este sistema se vuelve a la idea inicial del generador de marcha y espera. Se pretende variar internamente la velocidad del conjunto de registros de desplazamiento que generan la secuencia en función de los valores de la secuencia de salida de un segundo sistema de LFSR controlador. Es la misma idea del control de marcha y espera aunque más evolucionada. El mayor inconveniente de esta opción está en la exigencia de una mayor velocidad física del sistema generador, lo cual encarecerá el sistema. Actualmente, y para ciertas aplicaciones como la transmisión de datos a través de *modems* de 9.600 bits por segundo, existen circuitos integrados muy baratos que permiten este margen de sobrevelocidades. Dado un LFSR, si conocemos su longitud, su polinomio de realimentación, las conexiones no lineales y el estado inicial lo tendremos totalmente determinado. A pesar de ello, podemos conseguir salidas diferentes del mismo mediante el uso de un reloj de velocidad *d* veces mayor que la velocidad de reloj del sistema. A *d* se le denomina el factor de velocidad. Los valores que podremos alcanzar para *d* estarán limitados en cada momento por las consideraciones tecnológicas y económicas.

Fig. 3.17 Variador de la velocidad del reloj

Si se observa un LFSR con polinomio de realimentación primitivo y periodo *P* cada *d* intervalos de reloj interno, tal como muestra la figura 3.17, la secuencia de salida será una secuencia producible por un LFSR con periodo $P/\text{mcd}(d,P)$. Un caso interesante es el que cumple que $\text{mcd}(d,P)=1$, para el cual la secuencia observada cada *d* ciclos de reloj en una determinada celda de memoria es el contenido desplazado *d* fases del contenido de la celda contigua debido a la diferencia de velocidades. Esto ofrece la posibilidad de simular otro LFSR con fases muy separadas a partir de un LFSR controlado a una velocidad *d* adecuada.

A) Secuencias autodecimadas

Tal como se vio en el apartado anterior, al proceso de autoavance del reloj en función del valor del bit de salida de la propia secuencia lo llamamos decimación del registro de desplazamiento. La decimación usual por un valor constante *d* consiste en tomar muestras de la secuencia de salida a una

velocidad *d* veces menor que la de cambio de estado interno. Un nuevo tipo de decimación llamada decimación-[*d*,*k*] es la que se obtiene de avanzar *d* estados internos al sistema generador cuando se obtiene un cero binario de salida del sistema controlador, y avanzarlo *k* estados internos cuando tengamos un uno [RUE88], tal como muestra la figura 3.18.

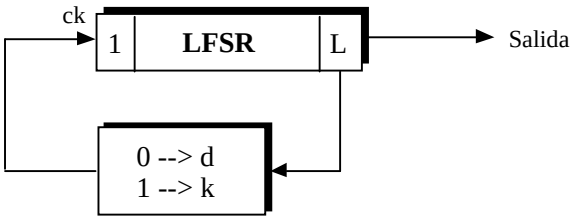


Fig. 3.18 Generador de secuencias autodecimadas

La decimación [*d*,*k*] no es pues una decimación por un valor constante, sino que se realiza por una función dependiente de los dígitos previos de la secuencia del registro del desplazamiento, y como resultado obtendremos una secuencia [*d*,*k*]-autodecimada. La aplicación de esta operación sobre registros cuyas salidas sean de longitud máxima y cuyos grados sean los mismos no hará variar las propiedades estadísticas de distribución de bits en la secuencia ni su periodo. Veamos algunos resultados en forma de teoremas. Para ello, consideremos el LFSR de la figura 3.18 con *L* celdas y periodo 2^L-1. Además debemos asumir que $0 < d,k < 2^L$-1, ya que si no, no sería posible reducir *d* o *k* mediante la operación mod2^L-1.

TEOREMA 1: Una secuencia de longitud máxima obtenida a partir de un LFSR de *L* celdas autodecimada [*d*,*k*] tendrá un periodo T_L =(2/3(2^L-1) cuando [d,k] = *g*(1,2) mod 2^L-1, si *g* un entero que cumple la condición mdc(*g*,2^L-1)=1. Observar que aún en el mejor caso posible, habrá una pérdida importante del 33% en la longitud de la secuencia de salida.

TEOREMA 2: La frecuencia absoluta de unos $N_L(1)$ en un periodo de una secuencia de longitud máxima generada por un registro de desplazamiento de orden *L* y autodecimada [*d*,*k*] es $N_L(1) = 1/3(2^L\text{-}1)$, con las mismas restricciones del teorema 1. Si *L* es par, la distribución es perfectamente balanceada en cuanto al número absoluto de unos y ceros de salida ($N_L(1) = N_L(0) = T_L/2$).

TEOREMA 3: Considerando las mismas restricciones que antes, la frecuencia absoluta de parejas de bits posibles $N_L(b_1,b_2)$ en un periodo completo de la secuencia de longitud máxima autodecimada [*d*,*k*] estará limitada por:

$$N_L^0 \le N_L(b_1, b_2) \le N_L^0 + 2$$

donde:

$N_L^0(00) = (1/6(2^L\text{-}1))\text{-}1$
$N_L^0(01) = 1/6(2^L\text{-}1)$
$N_L^0(10) = 1/6(2^L\text{-}4)$
$N_L^0(11) = 1/6(2^L\text{-}4)$

Para valores de L grandes la diferencia entre las distribuciones puede considerarse como nula.

TEOREMA 4: El periodo T_L de una secuencia de longitud máxima generada por un registro de desplazamiento de orden L y autodecimada $[d, d+1]$ está limitado por $T_L \leq (3/4(2^L-1))$.

Cuando se eligen adecuadamente los valores de d y k se pueden alcanzar propiedades estadísticas semejantes a las de las m-secuencias y, por tanto, cercanas a las distribuciones aleatorias ideales, ya que se obtiene un gran periodo, distribuciones de grupos de ceros y unos balanceados, así como similar número de ceros y unos totales en la secuencia y buena autocorrelación.

Sin embargo, la principal ventaja de este método está en la altísima (casi ideal) complejidad lineal alcanzada al autodecimar $[d,k]$ la m-secuencia.

3.4.4 Secuencias PN^2

Un caso especial de secuencias obtenidas mediante técnicas de control de reloj son las que se obtienen cuando los unos producidos por un generador de secuencias pseudoaleatorias (PN) se usan para activar el reloj de otro generador de secuencias pseudoaleatorias (PN). Las secuencias así obtenidas se las denominan PN^2, y este proceso se puede iterar poniendo generadores PN en cascada de forma que la salida de cada uno de ellos ataque al reloj del siguiente para obtener secuencias PN^k (si hay k generadores PN en cascada). Las secuencias PN^k [TRE74] serán secuencias binarias con una probabilidad para los unos de $(1/2)^k$.

La figura 3.19 muestra el caso para $k = 3$, es decir, es un generador PN^3. Suponemos que todos los generadores tienen periodo P aunque pueden generar secuencias diferentes. El primer generador PN tendrá un reloj constante (el reloj del sistema), mientras que el segundo generador PN recibirá pulso de reloj sólo cuando el primer generador PN produzca un 1. De este modo, la secuencia PN^2 contendrá un 1 cuando la salida del segundo generador PN realice una transición, y un cero en el caso contrario.

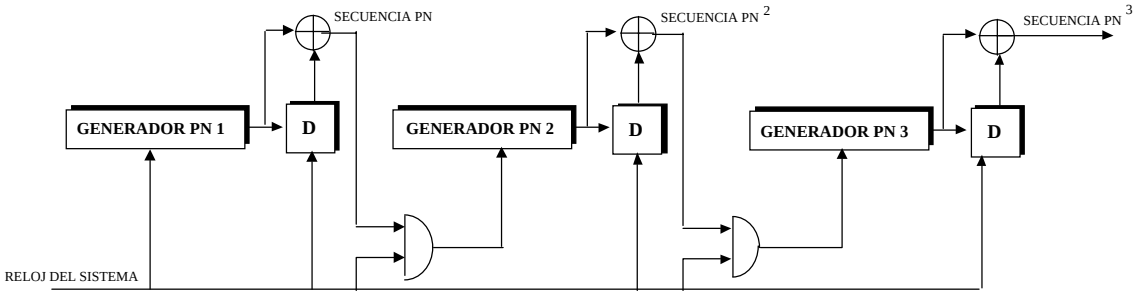


Fig. 3.19 Generador de secuencias PN^2 y PN^3

Veamos algunas propiedades de las secuencias PN^2, si ambos generadores PN son LFSRs de L celdas con polinomio de realimentación primitivo:

1) Si el periodo de la secuencia PN asociada es $P = 2^L-1$, el periodo de la secuencia PN^2 será P^2. En dicho periodo el segundo generador PN realizará $(P+1)/2$ ciclos.

2) Un periodo de una secuencia PN^2 contiene $[(P+1)/2]^2$ unos. Esto implica que la densidad de unos será $(1/4)[(P+1)/P]^2$ con lo que se aproximará a 1/4 cuando el periodo P crezca.

3) La autocorrelación normalizada fuera de fase no podrá exceder nunca 1/2 o ser menor que 1/4 y valdrá 1/4 para todos los desplazamientos entre 0 y $2P-2L$ excepto para P.

3.5 Secuencias de De Bruijn

Para acabar este capítulo, vamos a ver un tipo de secuencias llamadas secuencias de De Bruijn [ARA81] que, al igual que las *m*-secuencias, se usan como secuencias básicas para ser combinadas mediante funciones combinadoras. Las secuencias de De Bruijn son secuencias periódicas de periodo $P = 2^m$ en las que cada *m*-tupla aparece con igual probabilidad. Por tanto, son las secuencias más largas que se pueden generar con un registro de desplazamiento de *m* celdas. Además, en un periodo de dichas secuencias, hay el mismo número de ceros que de unos y presentan la máxima complejidad lineal posible para su longitud [CHA82].

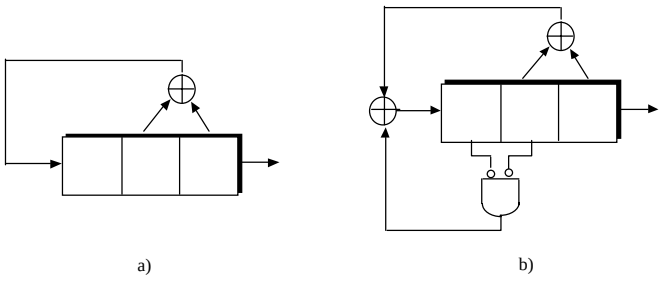


Fig. 3.20 LFSR (a) y generador de De Bruijn (b)

Existen algoritmos elegantes para generar este tipo de secuencias. Sin embargo, hay que tener en cuenta que, a medida que crece la elegancia matemática, la estructura de la secuencia se va volviendo más regular con lo que la aleatoriedad va decreciendo.

Las secuencias de De Bruijn son muy útiles en aplicaciones en las que cada *m*-tupla binaria debe aparecer dentro de un cierto retardo acotado, tal como el testeo lógico. Sin embargo, hay que tener en cuenta que la naturaleza altamente regular de dichas secuencias, que las hace muy útiles para una aplicación como la anterior, las vuelve poco eficientes para ser utilizadas como secuencias verdaderamente aleatorias.

La forma más simple de construir este tipo de secuencias consiste en añadir un cero adicional a la subsecuencia de *m*-1 ceros sucesivos de una *m*-secuencia de periodo 2^m-1, generada por un registro de desplazamiento con realimentación lineal con polinomio de realimentación primitivo de grado *m* (*m*

celdas) sobre GF(2). Este cero extra se inserta en la secuencia aplicando una función NOR al contenido de las *m*-1 primeras celdas del generador y añadiendo la salida (módulo 2) al canal de realimentación. La figura 3.20a muestra un generador de *m*-secuencias, y la figura 3.20b muestra una modificación sobre este generador para producir una secuencia de De Bruijn. Tal como se vio en el apartado 3.1.2, el número de distintas *m*-secuencias de periodo 2^m-1 viene determinado por $\phi(2^m - 1)/m$ (donde $\phi(x)$ es el número de enteros menores que x y mútuamente primos con él). Este procedimiento permitiría generar $\phi(2^m - 1)/m$ secuencias diferentes de De Bruijn de periodo 2^m.

3.6 Tabla de polinomios irreducibles sobre GF(2)

La tabla 3.4 muestra un conjunto de polinomios irreducibles de grados entre 2 y 17 que pueden ser empleados para realizar la realimentación en registros de desplazamiento de forma que se obtenga de ellos el máximo periodo.

Hay que notar que, para cada grado, la tabla de polinomios irreducibles que se muestra en este apartado no es exhaustiva y se pueden encontrar más polinomios en las tablas que da T. Herlestam [HER82].

Los polinomios que se pueden ver en la tabla 3.3 vienen expresados en forma octal, de forma que cada dígito representa 3 dígitos binarios según los siguientes códigos:

$$\begin{aligned} 0 &\rightarrow 000 \\ 1 &\rightarrow 001 \\ 2 &\rightarrow 010 \\ 3 &\rightarrow 011 \\ 4 &\rightarrow 100 \\ 5 &\rightarrow 101 \\ 6 &\rightarrow 110 \\ 7 &\rightarrow 111 \end{aligned}$$

Los dígitos binarios representan los coeficientes del polinomio de realimentación, donde el coeficiente de mayor orden es el de la izquierda.

Por ejemplo, un polinomio de grado 10 que estuviese representado por mediante la forma octal 3525, se podría expresar en la forma binaria como 011101010101 que, puesto a su vez en forma polinómica sería:

$$P(x) = x^{10} + x^9 + x^8 + x^6 + x^4 + x^2 + 1$$

Tabla 3.3

E,F,G,H	Polinomio primitivo
A,B,E,F	Las raíces son linealmente dependientes
C,D,G,H	Las raíces son linealmente independientes

Tabla 3.4 Polinomios irreducibles hasta grado 17

Grado 2	7H					
Grado 3	13F					
Grado 4	23F					
Grado 5	45E	75G	67H			
Grado 6	103F	127B	147H	111A	015	155E
Grado 7	211E 203F	217E 313H	235E 345G	367H	277E	325G
Grado 8	435E 537F	551E 703H	747H 471A	435F 763D	545E 727D	543F 433B
Grado 9	1021E 1605G	1461G 1743H	1055E 1617H	1167F 1553H	1541E 1773G	1333F 1267E
Grado 10	2011E 3023H	2415E 3543F	3771G 2725G	2157F 3471G	3515G 2707E	2773F 3323H
Grado 11	4005E 5235E	6015G 7431G	7413H 5607F	4143F 4731E	6233H 6037H	7237H 5253F
Grado 12	10123F 14227H	15647E 13131E	16533H 15053H	13565E 14357F	17121G 15413H	16521E 10605E
Grado 13	20033F 22471E	23261E 30465G	24623F 25245E	30741G 33357H	34517H 22233F	22631E 25633F
Grado 14	42103F 65153H	43333E 73647H	51761E 70553F	40503F 67101G	77141G 73143F	67517H 76047H
Grado 15	100003F 140573H	102043F 150633H	177775E 116637F	161007H 175515G	142305G 115537E	154463H 162455G
Grado 16	210013F 211473F	234313F 337553H	307107H 245367F	201735E 356057H	305667H 355507H	346355G 251741E
Grado 17	400011E 611073H	400017F 477225E	600013H 603777H	561175E 437631E	547163F 667401G	755723H 512211E

El término 1 de la derecha representa la línea que sirve de realimentación hacia el registro de desplazamiento. Por otra parte, la tabla 3.3 muestra el significado de las letras que acompañan a los polinomios de la tabla 3.4.

3.7 Problemas

PROBLEMA 3.1 Dado un LFSR de $L = 11$ celdas con polinomio de realimentación primitivo:

a) Determinar su periodo.
b) Calcular su complejidad lineal.
c) ¿Se podrá garantizar este periodo si se sustituye el polinomio primitivo por otro irreducible

no primitivo?
d) ¿Cuántos ceros habrá en un periodo?¿Y unos?
e) ¿Cuál será el valor fuera de fase de la función de autocorrelación?

PROBLEMA 3.2 Expresar la siguiente función binaria no lineal en la forma algebraica normal (ANF):

$$f(x_1, x_2, x_3) = x_1 + x_2 \cdot x_3 + x_1 \cdot x_2 \cdot x_3$$

donde + es la operación binaria OR y · es la operación binaria AND.

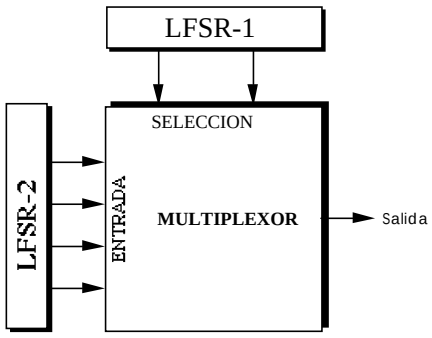
PROBLEMA 3.3 Dado el LFSR del problema 3.1, al que se le añade una función no lineal consistente en el producto de 3 de sus celdas determinar:

a) La máxima complejidad lineal que se podrá obtener en su secuencia de salida.
b) La complejidad mínima que tendrá la secuencia de salida si las 3 celdas que se toman como entradas a la función no lineal están equiespaciadas.

PROBLEMA 3.4 Dados dos LFSRs de L_1= 3 y L_2= 5 celdas respectivamente y polinomios de realimentación primitivos determinar el periodo y la complejidad lineal de la secuencia que se obtendrá si se combinan:

a) Haciendo el producto binario (AND) de ambas.
b) Haciendo la suma binaria (XOR) de ambas.

PROBLEMA 3.5 Determinar el periodo y la complejidad lineal que se obtendrá de la secuencia de salida de la siguiente estructura si el LFSR-1 tiene L_1= 5 celdas y el LFSR-2 tiene L_2= 11 celdas y ambos tienen polinomios primitivos:



PROBLEMA 3.6 ¿Qué diferencia habrá en el periodo y la complejidad lineal de la secuencia obtenida a partir de la estructura anterior si en un caso se usa un multiplexor con 2 líneas de seleción y 4 de entrada, y en otro caso se utiliza un multiplexor de 3 líneas de selección y 8 líneas de entrada?

PROBLEMA 3.7 ¿Qué periodo mostraría una estructura constituída por una báscula *JK* que combinase los mismos registros de desplazamiento del problema 3.5?

PROBLEMA 3.8 Si se realiza una [1,2]-decimación sobre una *m*-secuencia de periodo 65535:

a) ¿Cuál será el nuevo periodo de la secuencia resultante?
b) ¿Cuál será la frecuencia absoluta de los unos en la secuencia de salida?
c) Delimitar las frecuencias absolutas de las parejas 00, 01, 10 y 11.

PROBLEMA 3.9 Se construye un generador PN^2 a partir de dos LFSRs, ambos de 3 celdas y con polinomio de realimentación primitivo aunque diferentes. Determinar:

a) El periodo de la secuencia PN^2 obtenida.
b) El número de ceros y unos en dicho periodo.

3.8 Bibliografía

[ARA81] ARAZI, B. *On the Synthesis of De-Bruijn Sequences*. Information and Control, nº 49, pp.81-90, 1981.

[BAR58] BARSALL-RISTENBATT. *Introduction to Linear Shift Register Sequences*. University of Michigan, IT-90, 1958.

[BEA85] BEALE, M.; LAU, S. M. S. *On a Class of De Bruijn Sequences for Stream Ciphers*. IEEE Int. Info. Theory, Brighton, 1985.

[BEN76] BENJAUTHRIT, B.; REED, I. S. *Galois Switching Functions and Their Applications*. IEEE Trans. on Computers, vol. C-25, Nº 1, Jan 1976.

[BER68] BERLEKAMP, E. R. *Algebraic Coding theory*. McGraw-Hill, New-York, 1968.

[BET85] BETH, T.; PIPER, F. C. *The Stop & Go Generator*. Advances in Cryptology. Proceedings of Eurocrypt 84. Lecture Notes in Computer Science 209, pp. 88-92, 1985.

[BETHT] BETH, T.; COOK, B. M.; GOLLMANN, D. *Architectures for Exponentiation in GF(2).*

[CHA82] CHAN, A. H.; GAMES, R. A.; KEY, E. L. *On the Complexities of the De Bruijn Sequences*. J. Combin. Theory A, Vol. 33, pp. 233-246, 1982.

[CHA88] CHAMBERS, W. G. Clock Controlled shift Registers in Binary Sequence Generators. IEE Proc. E, 1988, 135, pp. 17-24.

[CHA88] CHAMBERS, W. G.; GOLLMANN, D. *Generators for Sequences with Near Maximal Linear Equivalence*. IE Proc. E, 1988, 135, pp. 7-69.

[CUMMI] CUMMINGS, L. J.; WIEDEMANN, D. *Embedded De Bruijn Sequences*. Discrete Mathematics.

[DAI90] DAI, Z. D.; BETH, T.; GOLLMANN, D. *Lower Bounds for the Linear complexity of Binary Sequences Derived form Sequences Over Residue Rings*. Proceedings of Eurocrypt, 1990.

[DAIHU] DAI, Z. D.; HUANG, M. Q. *A Criterion for Primitiveness fo Polynomials over Z/(2d)n*.

[DAIZD] DAI, Z. D. *Binary Sequences Derived from Maximal Length Linear Sequences Over Integral Residue Rings*.Proceedings of the Workshop on Stream Ciphers

[DOW69] DOWLING, T. A. *Irreducible Polynomials, Synchronisation Codes, Primitive Necklaces and the Cyclotomic Algebra*. Ed. Chapel Hill. Combinatorial Mathematics and its Applications. University of North Carolina Press, 1969, pp. 358-370.

[DYN87] DYNKIN, V. N.; MUSAELYN, S. .S. *Algorithm for the Generation of Non-linear De Bruijn Sequences*. Automation and Remote Control, Vol. 48, pp.125-129, 1987.

[ELD58] ELDERT, C.; GRAY, H. M.; RUBINOFF, M. *Shifting Counters*. AIEE Trans. (Commun. Electron), Vol. 7, pp. 70-74, March 1958.

[ELS59] ELSPAS, B. *Theory of Autonomous Linear Sequentioal Networks.* IRE Trans. Circuit Theory, Vol. CT-6, pp. 45-60, March 1959.

[ETZ84] ETZION, T.; LEMPEL, A. *Algorithm for the Generation of Full-Length Shift Register Sequences.* IEEE Transactions on Information theory, 30, pp. 480-484, 1984.

[ETZ84] ETZION, T.; LEMPEL, A. *Construction of De Bruijn Sequences of Minimal Complexity*. IEEE Transactions Info. Theory, Vol. IT-30, pp. 705-709, 1984.

[FOR57] FORD, L. JR. *A Cyclic Arrangements of m-tuples*. RAND Corporation, Santa Monica, Calif, Rept. P-1071, Aprl, 1957.

[FRE82] FREDRICKSEN, H. *A Survey of Full Length Nonlinear Shift Register Cycle Algorithms.* SIAM Review, Vol. 24, Nº 2, April 1982.

[GAM83] GAMES, R. A.; CHAN, A. H. *A Fast Algorithm for Determining the Complexity of a De Bruijn Sequence with Period* 2^n. IEEE Transactions Info. Theory, IT-29, pp.705-709, 1983.

[GOL59] GOLOMB, S. W. *On the Classification of Boolean functions.* IEE Trans. on Inform theory, Vol. IT-5, pp. 176-186, May. 1959.

[GOL59] GOLOMB, S. W.; WELCH, L. R.; GOLDSTEIN. *Cycles From Nonlinear Shift Registers*. Jet Propulsion Lab, California Institute of Technology, Pasadena, Calif, Prog. Rept. 20-389, August 1959.

[GOL67] GOLOMB, S. W. *Shift Register Sequences*. San Francisco, Holden-Day, 1967.

[GOL85] GOLLMANN, D. Pseudo Random Properties of Cascade Connections of Clock Controlled Shift Registers. Advances in Cryptology. Proceedings of Eurocrypt 84. Lecture Notes in Computer Science 209, pp. 93-98, 1985.

[GOL89] GOLIC, D. J. *On the Linear Complexity of Functions of Periodic GF(q) Sequences*. IEEE Trans on Inform theory, vol. IT-35, Nº1, Jan, 1989.

[GOL89] GOLLMANN, D.; CHAMBERS, W. G. *Clock-Controlled Shift Registers: A Review*. IEEE Journal on Selected Areas in Communications, Vol. 7, Nº 4, May 1989.

[GRO71] GROTH, E. J. *Generation of Binary Sequences with Controllable Complexity*. IEEE Trans. Inform. Theory, Vol. IT-17, pp.288-296, 1971.

[HER82] HERLESTAM, T. *On Using Prime Polynomials on Crypto Generators*. Proc. Workshop on Cryptography, Springer-Verlag Lecture notes in computer Science, Nº 149, NY, 1982.

[INF68] *Theory of Transformation Groups of Polynomials Over GF(2) with Applications to Linear Shift Register Sequences*. Inform. Sci. Vol. 1, Nº 1, Dec. 1968, pp. 87-109.

[KAR76] KAROVSKY, M. G. *Finite Orthogonal Series in the Design of Digital Devices*. New York and Jerusalem: Wiley and IUP, 1976.

[KEY76] KEY, E. L. *An Analysis of the Structure and Complexity of Non-linear Binary Sequence Generators.* IEEE Trans. Inform theory, Vol. IT-22, pp. 732-736, Nov. 1976.

[LEA68] LEACH, E. B. *Regular Sequences and Frequency Distribution*. Proc. Amer. Math. Soc., Vol. 11, august, 1968.

[LEM70] LEMPEL, A. *High Speed Generation of Maximal-Length Sequences.* IEEE Trans. on Computers, Vol. C-19, Feb. 1970.

[LEM71] LEMPEL, A. *Analysis and Synthesis of Polynomial and Sequences Over GF(2)*. IEEE Transactions on Information theory, vol. IT-17, Nº 3, May 1971.

[LEM71] LEMPEL, A. *M-ary Closed Sequences*. J. Combinatorial Theory, March 1971.

[LIU64] LIU, R.; MASSEY, J. *Monotone Feedback Shift Registers*. Proc. 2nd Allerton Conference on circuit and System Theory, Univ. of Illinois, Urbana, 1964, pp. 860-874.

[MAG63] MAGLEBY, K. B. *The Systhesis of Nonlinear Feedbak Shift Registers*. TR 6207-1, Standford Elec. Lab, Standford, CA, 1963.

[MAS69] MASSEY, J. L. *Shift Register sysnthesis and DCH Decoding*. IEEE TRans. on Inform theory, Vol. IT-15, Nº 1, Jan. 1969.

[PAR89] PARK, W. J.; KOMO, J.J. *Relationships Between m-Sequences Over GF(q) and GF(q^m)*. IEE Trans. on Inform theory, Vol. 35, Nº1, Jan. 1989.

[RUE87] RUEPPLE, R. A.; STAFFELBACH, O. J. *Products of Linear Recurring Sequences with Maximum complexity*. IEEE Trans. Inform. Theory, Vol. IT-33, pp. 124-131, 1987.

[RUE88] RUEPPLE, R. A. *When Shift Registers Clock Themselves*. Proc. Eurocrypt 87. Lecture notes in computer Science, 309, Springer Verlag, 1988, pp. 53-64.

[SEL66] SELMER, E. S. *Linear Recurrence Relations Over finite fields*. Dep. math. Univ. Bergen, Norway, 1966.

[SIE84] SIEGENTHALER, T. *Correlation Inmunity of Nonlinear Combining Functions for Cryptographic Applications*. IEEE Trans Info Theory, IT-30, pp. 776-780, 1984.

[SME86] SMEETS, B. *A Note on Sequences Generated by Clock Controlled Shift Registers*. Advances in Cryptology: eurocrypt 85. Lecture Notes in Computer Science, 219, pp. 142-148, 1986.

[SUR78] SURBOCK, F.; WEINRICHTER, H. *Interlacing Properties of Shift-Registers Sequences With Generator Polynomials Irreducible Over GF(p)*. IEEE Trans. on Inform. theory, vol. IT-24, pp. 386-389, 1978.

[TSA64] TSAO, S. H. *Generation of Delayed Replicas of Maximal-Length Linear Binary Sequences*. Proc. IEE, Vol. 111, pp. 1803-1806, 1964.

[TRE74] TRETTER, S. A. *Properties of PN^2 Sequences*. IEEE Trans. on Inform. Theo, March 1974.

[WAR33] WARD, M. *The Aritmethical Theory of Linear Recurring Series*. Transactions of the American Mathematical Society, 35, pp. 600-628, July 1933.

[YOE61] YOELI, M. *Nonlinear Feedback Shift Registers*. IBM Develment Lab, Poughkeesie, N.Y. Tech Rept, TR00.809,September, 1961.

[ZIE59] ZIERLER, N. *Linear Recurring Sequences*. J. Soc. Ind. Appl. Math. Vol 7., 1959.

4 Algoritmos de generación por *hardware*

4.0 Introducción

Si bien en el capítulo 2 se describieron una serie de técnicas para generar secuencias de números aleatorios mediante ordenador, en muchas aplicaciones usadas en telecomunicaciones es atractivo, e incluso imprescindible, la generación de este tipo de secuencias prescindiendo de éstos y utilizando dispositivos electrónicos de tipo común. Las especificaciones que se piden a las secuencias generadas son las mismas que en el caso anterior. Sin embargo, la eficiencia, la rapidez de obtención de los bits y la menor complejidad estructural de los dispositivos implicados adquieren una mayor importancia de la que tendrían en el caso de que las secuencias se obtuviesen a partir de un programa ejecutado sobre un ordenador.

En este capítulo se van a ver una serie de dispositivos que, con técnicas descritas en el capítulo 3 y otras nuevas, generarán secuencias pseudoaleatorias empleando para ello dispositivos electrónicos comúnmente utilizados y conocidos. Vamos a dividir estos generadores en tres grupos diferentes. Al primer grupo pertenecerán aquellos en los que la generación se basa en combinar generadores pseudoaleatorios básicos, como pueden ser los registros de desplazamiento con realimentación lineal (LFSR) mediante funciones combinadoras y añadiendo funciones no lineales según se requiera.

Al segundo grupo pertenecerán todos aquellos que se basan en el empleo de técnicas de control de reloj. La teoría básica de estos dos primeros grupos se vio en el capítulo 3. Finalmente habrá un tercer grupo de generadores que por su naturaleza no emplean ninguna de las técnicas de los anteriores y, por tanto, se verán como generadores especiales.

4.1 Generadores basados en funciones combinadoras

4.1.1 El generador de Jennings

Este es un esquema propuesto por S.M. Jennings en 1980 [JEN80] y que usa un multiplexor para combinar las secuencias de salida de dos registros de desplazamiento con realimentación lineal (LFSR-1 de *m* celdas y LFSR-2 de *n* celdas) tal como se muestra en la figura 4.1 [JEN82].

El generador produce la señal de salida $z(t)$, $t \geq 0$ de la siguiente forma: se fija un entero positivo h que cumpla que $h \leq \min(m, \log_2 n)$ y una función de transformación $0 \leq i_0 < i_1 < ... < i_{h-1} \leq m - 1$, que actúe sobre h celdas del LFSR-1 de forma que en cada momento, para $t \geq 0$, forme el número: $u(t) = a(t+i_0) + a(t+i_1)2 + ... + a(t+i_{h-1})2^{h-1}$ a partir de dichas celdas y lo transforme en el número $ß(u(t)) = s_0(t) + s_1(t)2 + ... + s_{k-1}(t)2^{k-1}$, donde $k = \lfloor log_2 n \rfloor$.

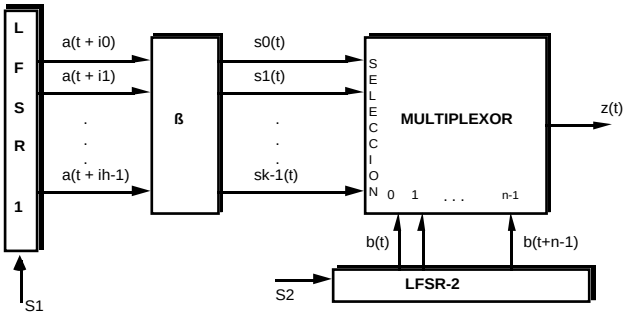


Fig. 4.1 Generador de Jennings

Realiza, pues, una aplicación inyectiva $ß : \{0, 1, ..., 2^h-1\} \rightarrow \{0,1, ..., n-1\}$. La familia de transformaciones ß posibles es enormemente grande. Con esto, la señal de salida del generador se define como:

$$z(t) = b[t + ß(u(t))]$$

Jennings [JEN80] demuestra que si ambos LFSR tienen polinomios primitivos y sus órdenes (número de celdas) son mutuamente primos entre sí (mcd(m,n) = 1), la secuencia de salida tendrá un periodo:

$$T = (2^m-1)(2^n-1)$$

y la complejidad lineal estará acotada superiormente por:

$$\Lambda(z) \leq n\left(1 + \sum_{i=1}^{h} C_i^m\right)$$

donde se dará la igualdad si las posiciones de las celdas del LFSR-1, que se eligen como entrada a la función de transformación ß, están equiespaciadas. La semilla de este generador la constituye el contenido inicial de las celdas de ambos LFSR.

Una mirada más profunda a la definición de la salida $z(t)$ revela que, independientemente de cuál sea la función de transformación ß, la señal se puede expresar como una combinación lineal de las $b(t)$, con los coeficientes dependiendo principalmente de S_1. Este será el talón de Aquiles si este

generador se emplea en aplicaciones como la criptografía, ya que se le puede aplicar un test de consistencia lineal (LCT) [ZEN89] para averiguar los parámetros del generador y poder determinar así completamente la secuencia de salida. Esto se verá en el capítulo 7. Si los polinomios de realimentación son conocidos, un posible atacante podría, conociendo tan sólo una porción de $N \geq m+n2^h$ bits de la secuencia $z(t)$ de salida, determinar completamente la estructura del generador. Para ello, bastaría que aplicase 2^{m+h} tests de consistencia y tan sólo le quedaría obtener la semilla S_1 utilizando una búsqueda exhaustiva (consistente en probar todas las posibilidades).

Este generador está propuesto por la EBU (European Broadcasting Union) para realizar el cifrado en el estándar de transmisión de televisión MAC. Una descripción de cómo se realiza el cifrado en este sistema se puede ver en el capítulo 7, mientras que en el último capítulo se realiza un análisis de este generador.

4.1.2 El generador de Geffe

Otra forma muy simple de combinar registros de desplazamiento realimentados linealmente mediante un multiplexor es la que muestra la figura 4.2 [GEF73] En esta estructura, las secuencias producidas por dos LFSR se usan como entradas de un multiplexor y, para introducir más aleatoriedad, se emplea la secuencia producida por otro multiplexor en la entrada de selección para decidir cuál de los dos LFSR de entrada es el que va a dar su bit a la salida. Esta estructura también se puede poner en la forma que muestra la figura 4.3.

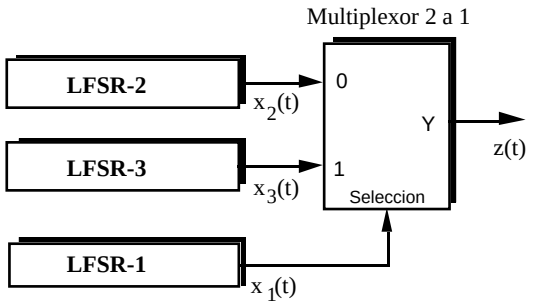


Fig. 4.2 Generador de Geffe

Si llamamos $x_2(t)$ y $x_3(t)$ a las señales binarias de entrada del multiplexor, y $x_1(t)$ a la señal que ataca a la entrada de selección, la salida del generador en el instante t es:

$$z(t) = x_1(t)x_3(t) \oplus \overline{x_1(t)}x_2(t) = x_2(t) \oplus x_1(t)(x_2(t) \oplus x_3(t))$$

o también:

$$z(t) = x_3(t) \oplus \overline{x_1(t)}(x_2(t) \oplus x_3(t))$$

Si los registros de desplazamiento con realimentación lineal LFSR-1, LFSR-2 y LFSR-3 tienen grados n_1, n_2 y n_3 respectivamente, y polinomio de realimentación primitivo, el periodo de la secuencia de salida z(t) de este generador será:

$$T = mcm(2^{n_1} - 1, 2^{n_2} - 1, 2^{n_3} - 1)$$

y la complejidad lineal:

$$\Lambda = n_3 n_1 + (n_1 + 1) n_2$$

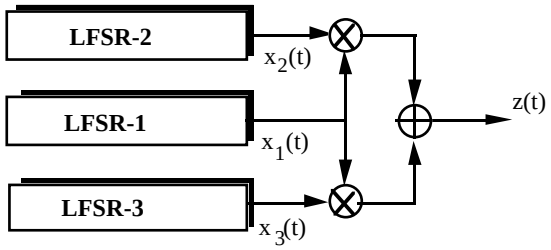


Fig. 4.3 Otra forma del generador de Geffe

Este generador no es aconsejable en ciertas aplicaciones como la criptografía, ya que presenta una debilidad debida a que existe una probabilidad de coincidencia entre la salida z(t) y la secuencia de entrada x_2(t) que es mayor que 1/2:

$$P = Prob(z(t) = x_2(t)) = Prob(x_1(t) = 0) + Prob(x_1(t) = 1)\, Prob(x_3(t) = x_2(t)) = \frac{1}{2} + \frac{1}{4}$$

y lo mismo ocurre entre z(t) y x_3(t) ya que

$$P = Prob(z(t) = x_3(t)) = \frac{1}{2} + \frac{1}{4}$$

y se puede estimar de igual modo que la anterior. Además, si un posible atacante conociera los polinomios de realimentación de los registros de desplazamiento y éstos fueran primitivos y de grado menor que un cierto valor n, podría romper fácilmente el generador empleando para ello un ataque basado en el método del síndrome lineal [ZEN89], y podría obtener el resto de la estructura del generador necesitando tan sólo una porción de la secuencia de salida de longitud $N = 37n$.

Para llevar a cabo este ataque tan sólo necesitaría realizar $896n$ operaciones binarias. Existen además otras técnicas basadas en los ataques por correlación que permiten criptoanalizar sistemas de cifrado en flujo basados en esta estructura. Tanto el método del síndrome lineal como los ataques por correlación para secuencias empleadas en sistemas criptográficos se tratarán con más detalle en el capítulo 7.

4.1.3 El generador de umbral

Este generador de secuencias pseudoaleatorias, propuesto por J.O Bruer en 1982 [BRU82], tiene una estructura basada en N registros de desplazamiento con realimentación lineal obtenida mediante polinomios primitivos combinados de la siguiente forma:

$$z_i = \begin{cases} 1 \quad si \quad \sum_{j=1}^{N} x_{ji} > N/2 \\ 0 \quad en \quad otro \quad caso \end{cases}$$

En cada instante t, la salida de los N registros de desplazamiento se suman, y el resultado de esta suma se pasa a través de un detector de umbral, tal como se ve en la figura 4.4. La salida del generador $z(t)$ será igual a 1 si el número de unos a la entrada excede de $N/2$, y 0 en caso contrario.

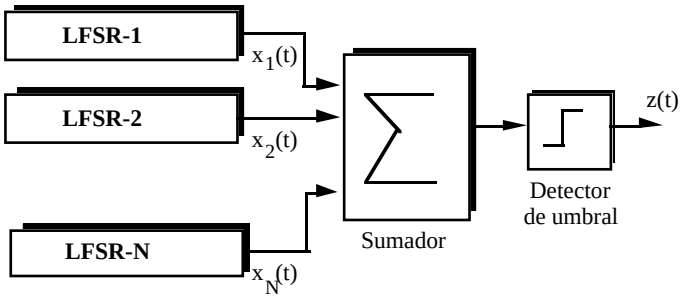


Fig. 4.4 Generador de umbral

La salida de este generador estará balanceada sólo si el número de LFSRs utilizado es impar. Veamos un ejemplo con $N = 3$ en la que la salida equivalente, puesta en notación algebraica normal (ANF) sea: $z_i = x_{1i}x_{2i} \oplus x_{1i}x_{3i} \oplus x_{2i}x_{3i}$.

Esta forma de representación viene determinada por la suma módulo 2 de todos los productos de segundo orden. Si el número de celdas de los 3 LFSRs son L_1, L_2 y L_3 respectivamente y sus periodos P_1, P_2 y P_3, el periodo P de la secuencia $z(t)$ y la complejidad lineal $\Lambda(z)$ serán:

$$P = P_1 \cdot P_2 \cdot P_3 \qquad \Lambda(z) = L_1L_2 + L_1L_3 + L_2L_3$$

Hay que tener en cuenta que existirá una cierta correlación entre la secuencia de salida $y(t)$ y cada una de las secuencias producidas por los LFSRs. Esto representa un inconveniente si se usa este generador para aplicaciones de cifrado en flujo, ya que puede ser aprovechada para criptoanalizar el sistema. Este generador no es aconsejable para aplicaciones criptográficas de cifrado en flujo, ya que presenta ciertas debilidades frente a ataques basados en la correlación.

4.1.4 El generador de Pless

Veamos en este apartado un esquema propuesto por V.S. Pless [PLE77] basado en usar básculas *J-K* como funciones combinadoras de secuencias generadas por registros de desplazamiento con realimentación lineal. Este tipo de estructura preserva las buenas propiedades de aleatoriedad de las secuencias producidas por los LFSR y, además, soluciona el problema de la linealidad inherente en estos dispositivos, que puede ser importante en ciertas aplicaciones como las criptográficas debido a la existencia de numerosos ataques que aprovechan la debilidad de su linealidad inherente.

La báscula *J-K* es bien conocida por su utilidad en circuitos digitales (ya estudiado en el capítulo anterior). Es un dispositivo de dos entradas (*J*,*K*) y dos salidas (donde una salida es la complementaria de la otra), y que opera de la siguiente forma: Consideremos un par ordenado (*J*,*K*) para representar la entrada. Un (0,0) deja la salida sin alterar, es decir, con el valor que tenía antes de cambiar la entrada. Un (1,1) a la entrada invierte el estado que tendría la salida en el instante anterior (de 0 a 1 o viceversa). Una entrada (0,1) produce un 0 a la salida y, finalmente, una entrada (1,0) produce un 1 a la salida.

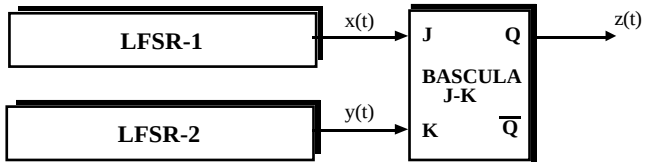


Fig. 4.5 Generador de Pless

Veamos una primera estructura basada en una sola báscula *J-K* que combina la salida de los LFSRs, tal como muestra la figura 4.5. De esta estructura se pueden hacer las siguientes afirmaciones:

Si el LFSR-1 produce una secuencia de periodo $P_1 \neq 1$, y el LFSR-2 produce una secuencia de periodo $P_2 \neq 1$, y se cumple que el mcd(P_1,P_2)=1, y que tanto P_1 como P_2 son impares, entonces el periodo de la secuencia que produce esta estructura P es $P = P_1P_2$. Esta estructura es muy fácil de implementar debido a la sencillez de todos los elementos que la componen y, además, presenta buenas propiedades en el caso de errores en la transmisión de la secuencia producida, ya que si este error se produce en un bit producido por el *flip-flop J-K*, se tratará de un único error que no afectará a los otros bits. Si se produce un error en el estado interno de la báscula *J-K*, éste afectará a todos los bits mientras la entrada (*J*,*K*) sea (0,0) o (1,1). Sin embargo, este error se corregirá tan pronto como la entrada pase a valer (1,0) o (0,1). Por tanto, tendremos un flujo de bits completamente incorrecto pero muy fácil de detectar o un flujo de bits completamnte correcto.

Sin embargo, esta estructura no preserva las buenas propiedades estadísticas que presentan las secuencias producidas por los LFSR que actúan de entrada en la báscula *J-K*. Si en la entrada de la bascula hay un 1, la probabilidad de que la siguiente salida sea un uno es menor que 1/2, y ocurre de forma similar cuando la salida es un cero.

Esto también es un problema si se pretende emplear esta estructura en aplicaciones de cifrado, ya que el sistema será más susceptible a ataques estadísticos que una secuencia realmente aleatoria.

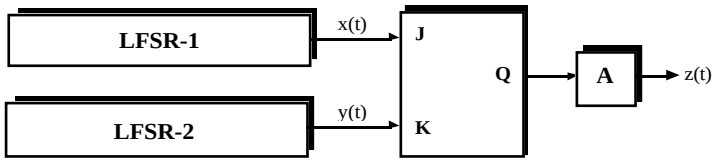


Fig. 4.6 Mejora sobre el generador de Pless

En el capítulo 3 se vio que la estructura de un LFSR de L celdas se puede determinar analizando $2L$ bits producidos a la salida. Este problema se mantiene con esta estructura, ya que si el LFSR-1 tiene m celdas y el LFSR-2 tiene n celdas, se puede determinar la estructura estudiando tan sólo $2n + 2m$ bits producidos por ella. El problema de la no aleatoriedad de la salida se puede paliar, en parte, añadiendo a la salida de la estructura anterior un dispositivo A que elimine bits de forma alternada, tal como muestra la figura 4.6. Para esta estructura, también se cumple que si el LFSR-1 tiene periodo impar P_1, y el LFSR-2 tiene periodo impar P_2, y el mcd(P_1,P_2) = 1, el periodo de la secuencia de salida será $P = P_1P_2$. Además, el alternador A restaura parte de las buenas propiedades estadísticas de los LFSRs de entrada, ya que 2 es primo relativo con P_1P_2 cuando P_1P_2 es impar. La desventaja de esta estructura es que emite un bit cada dos pulsos de reloj, por lo que el registro de desplazamiento debe operar al doble de la velocidad de la secuencia de entrada.

Tabla 4.1

LFSR	*Nº celdas*	*Periodo*	*Nº de elecciones*	*Factorización de P*
1	5	31	6	31
2	19	524287	27594	524287
3	7	127	18	127
4	17	131071	7710	131071
5	9	511	48	7*73
6	16	65635	2048	3*5*17*257
7	11	2047	176	23*89
8	13	8191	630	8191

Otra posible estructura es la que se muestra en la figura 4.7, donde se emplean 4 LFSRs, 3 básculas J-K y 3 alternadores. Si se asume que r_1, r_2, r_3 y r_4 son las celdas de los 4 LFSR respectivamente, para poder obtener los parámetros de la estructura se necesitarían al menos $2^{(2^{r_1}+r_2+2^{r_3}+r_4)}$ bits alternados. Este ya es un valor elevado para valores de r_1, r_2, r_3 y r_4 pares y moderadamente grandes. Esta estructura presenta cierto atractivo para usarla en aplicaciones de tipo criptográfico. Las desventajas de esta estructura son, que es difícil de simular mediante un ordenador, y no es posible ejecutarla en tiempo real. Como ventaja tiene que es muy fácil de implementar.

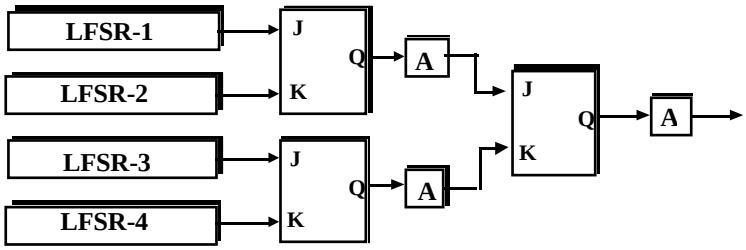


Fig. 4.7

Una última estructura presentada por Pless es la de la figura 4.8, que mantiene las propiedad de ser difícil de romper mediante cálculos lineales, que ya presentaba la estructura anterior, además de permitir su ejecución en tiempo real. El dispositivo con cajas numeradas de 1 a 4 es un contador cíclico y transmite el contenido de la caja $i + 1 \pmod 4$ a la derecha después de que se haya transmitido el contenido de la caja i. En esta estructura, si se usan M LFSR de forma que las longitudes de sus secuencias de salida sean impares y mutuamente primas dos a dos, el periodo P de la secuencia de salida será:

$$P = \sum_{i=1}^{M} P_i$$

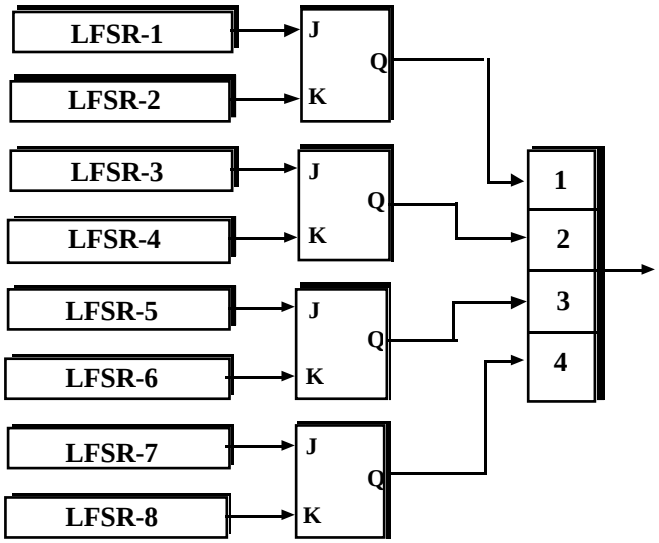


Fig. 4.8

La tabla 4.1 muestra una posible elección del número de celdas (r_i) para los $N = 8$ LFSR. Además, se eligen polinomios primitivos para todos ellos. De dicha tabla se puede observar que los

periodos de los 8 LFSR son impares y mutuamente primos dos a dos, por lo que el periodo de la estructura completa será su producto, que es mayor que 10^{28}.

El número de elecciones diferentes de polinomios primitivos que sean capaces de producir esos periodos para los 8 LFSRs se muestra en la cuarta columna de la tabla.

Para la estructura completa, el número de elecciones diferentes posibles de los 8 polinomios primitivos para los 8 LFSR será el producto de los valores de la quinta columna y será mayor que 2×10^{20}. Esto implica un número de posibilidades enorme que impediría, en el caso de que este generador se empleara, por ejemplo, en un sistema criptográfico, romper el criptosistema mediante el intento simple de todas las posibles combinaciones por parte de un usuario no autorizado.

Aún se podría hacer una variación en este último esquema que consistiría en la sustitución de los LFSR por contadores módulo *n* (formados por la interconexión de básculas *J-K*). Estos contadores podrían ser a su vez lineales o no. Estos últimos son más difíciles de determinar incluso que los LFSR, en el caso de un criptoanálisis contra un sistema de cifrado en flujo basado en esta estructura.

4.1.5 El generador de Rueppel

Este generador consiste en *N* registros de desplazamiento con realimentación lineal y un subsistema combinador no lineal consistente en un sumador de reales y memoria de longitud $\log_2 N + 1$, tal como muestra la figura 4.9. Fue propuesto por R.A. Rueppel [RUE86] en un intento de evitar el compromiso entre inmunidad a la correlación y la complejidad lineal que aparece en las funciones combinadoras sin memoria que se expuso en el capítulo 3. Este generador consiste en un subsistema conductor compuesto de *N* registros de desplazamiento con realimentación lineal y una función combinadora no lineal que incluye un sumador de reales y una memoria.

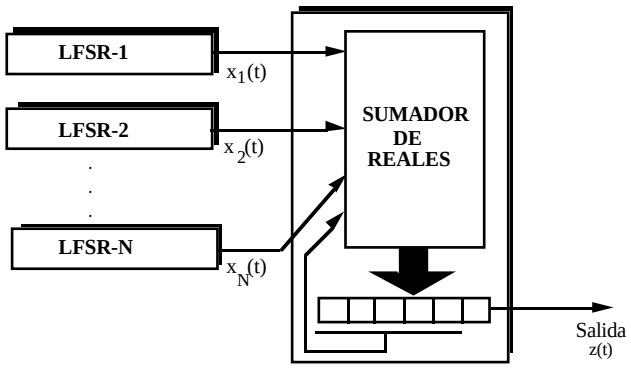


Fig. 4.9 Generador de Ruepple

El funcionamiento es el siguiente: En el instante 1, se suman como números reales (y no módulo 2) el primer bit de cada una de las *m*-secuencias producidas por los *N* registros de desplazamiento $x_i(1)$ (i = 1,..., N). El resultado es un número de log$N$ + 1 bits que se almacena en una

memoria interna. Una vez hecho esto, se desplaza el contenido de esta memoria un bit a la derecha, con lo que se obtiene el primer bit de la secuencia de salida a partir del bit menos significativo. En el instante 2, el contenido de la memoria y la nueva salida de los N LFSRs $x_i(2)$ (i = 1, ..., N), se vuelven a sumar (suma real) iterando así todo este procedimiento.

Si se eligen los polinomios de realimentación de los LFSR primitivos (entonces producen m-secuencias de periodo $2^{L_i} - 1$, donde L_i es el número de celdas y $1 \le i \le N$) y se impone la restricción de que los grados (número de celdas) sean relativamente primos uno a uno ($\forall i \neq j \quad 1 \le i,j \le N$, mcd($L_i$, L_j) = 1), la secuencia que produce el generador presenta las siguientes condiciones:

1) Gran periodo (el máximo): $P = \prod_{i=1}^{N} (2^{L_i} - 1)$.
2) Complejidad lineal aproximadamente la máxima:
$$\Lambda \approx \Lambda_{max} = \prod_{i=1}^{N} (2^{L_i} - 1)$$
3) Excelente propiedades estadísticas (los unos y los ceros son casi equiprobables).
4) Es posible conseguir inmunidad a la correlación de orden N - 1 (la máxima) debido a la memoria interna.

La inmunidad a la correlación, tal como se vio en el capítulo 3, define el grado de incorrelación (independencia estadística) entre las secuencias que producen los LFSR (subgeneradores) y la secuencia de salida. Si bien en ciertos casos esta propiedad no es relevante, en otros como la criptografía es de vital importancia cuando se emplean estructuras como ésta, basadas en varios subgeneradores, y una función no lineal que los combina. Si las secuencias no están incorreladas, un posible atacante puede aprovechar esta información mutua para realizar un ataque que permita obtener los parámetros de la estructura, incluída la clave secreta (que es el contenido inicial de las celdas de los N registros de desplazamiento).

Tal como vimos en el capítulo 3, el hecho de introducir memoria en la función combinadora no lineal permite conseguir simultáneamente máxima complejidad lineal y máxima inmunidad a la correlación.

4.1.6 Generador de Tatebayashi

En 1989, Tatebayashi, Matsuzaki y Newman [TAT89] propusieron una modificación sobre el generador de Ruepple (visto en el apartado anterior) para aplicaciones de cifrado en comunicaciones móviles. La estructura resultante se muestra en la figura 4.10, donde se ve que la modificación consiste en añadir un registro de control de longitud N (que llamamos REGCTRL). Con ello se consigue producir diferentes secuencias en lugar de una sola al usar diferentes vectores iniciales, por lo que se puede usar como clave durante una comunicación. Conviene recordar que al variar el contenido inicial de las celdas de los subgeneradores (registros de desplazamiento) lo que se consigue es modificar el punto en que se inicia la secuencia, pero ésta es siempre la misma. Al modificar el valor del registro REGCTRL se pueden conseguir 2^N secuencias diferentes.

Su contenido debe ser constante durante una transmisión y se puede considerar como un aumento de la clave secreta que hasta ahora estaba formada por el estado inicial de las celdas de los LFSR.

El funcionamiento de este generador es el siguiente: El REGCTRL tiene N bits de forma que el i-ésimo bit se suma módulo 2 (suma OR exclusiva) con la secuencia de salida $\{x_i\}$ del i-ésimo LFSR bit a bit, con lo que resulta la secuencia $\{y_i\}$. En el módulo sumador, las secuencias $\{y_i\}$ se suman como números reales, al igual que ocurría en el generador de Rueppel.

La adición de este registro de control no modifica las propiedades de periodo, complejidad y estadística que mostraba el generador de Rueppel.

También para este generador se eligen N LFSRs con grados mutuamente primos entre sí y polinomios de realimentación primitivos. Con estas condiciones, la secuencia de salida $z(t)$ tendrá las siguientes propiedades:

1) Periodo de salida máximo: $P = \prod_{i=1}^{N} (2^{L_i} - 1)$
2) Complejidad lineal casi máxima $\Lambda(z) \approx P_{max}$
3) Los unos y los ceros están balanceados.
4) Inmunidad a la correlación de máximo orden N - 1.

Se puede comprobar fácilmente la propiedad 3 mediante el siguiente razonamiento: Puesto que todos los LFSR ($LFSR_i$) del subsistema conductor generan m-secuencias $\{x_i\}$, ya se ha dicho que sus ceros y unos están balanceados. Las secuencias $\{y_i\}$ obtenidas también tendrán los unos y ceros balanceados, puesto que dichas secuencias son, o bien iguales a $\{x_i\}$, o bien iguales a $\{x_i\}$ pero complementadas.

Llamemos a $u_k(j)$ el bit menos significativo de la suma real de los N bits obtenidos de las secuencias $\{y_i\}$ (i = 1,...,N) más el acarreo (excepto el bit de la secuencia y_k) en el instante j:

$$u_k(j) = \sum_{i=1}^{N} y_i(j) + c_i(j)$$

El siguiente bit de la secuencia de salida del generador $z(t)$ es el bit menos significativo de la suma de todos los y_i y el acarreo $z(j) = y_k(j)+u_k(j)$. Entonces:

$$P(z=1) = P(y_k=1)P(u_k=0)+P(u_k=0)P(u_k=1) = 1/2P(u_k=0)+1/2P(u_k=1) = 1/2$$
$$P(z=0) = P(y_k=0)P(u_k=0)+P(y_k=1)P(u_k=1) = 1/2P(u_k=0)+1/2P(u_k=1) = 1/2$$

lo que demuestra el balanceo que existe entre los unos y los ceros en la secuencia $z(t)$ de salida.

También se puede demostrar el hecho de que el generador presente inmunidad a la correlación de máximo orden N - 1. Para ello, se define $u_k(j)$ de la misma forma que antes.

Puesto que en cada instante j, el nuevo bit de la secuencia de salida z(t) se obtiene mediante $z(j) = y_k(j) + u_k(j)$, entonces se cumple que $P(z=u_k) = P(y_k=0) = 1/2$ y que $P(z \neq u_k) = P(y_k=1) = 1/2$. Por tanto, la salida $z(t)$ estará incorrelada con $u_k(t)$, lo que implica que $z(t)$ presentará inmunidad a la correlación de orden N - 1, ya que u_k tiene N - 1 variables.

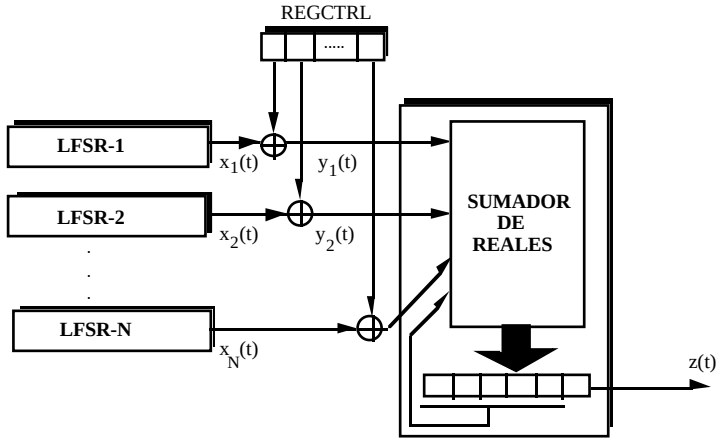


Fig. 4.10 Generador de Tatebayashi

4.2 Generadores basados en técnicas de control de reloj

Estos generadores se basan en controlar el reloj de un registro de desplazamiento con realimentación lineal (LFSR) mediante la salida de otro LFSR. Antes de ver estos generadores, recordemos algunos conceptos sobre técnicas de control de reloj [CHA88] vistos anteriormente en el capítulo 3. Consideremos un LFSR de control y un LFSR cuyo reloj es controlado por la salida del anterior, de polinomio de realimentación irreducible f(x) de grado n y orden N, y cuya salida llamamos $u(t)$. El funcionamiento de la técnica de control de reloj es el siguiente: Si después de la salida (t-1)-ésima del sistema, el LFSR de control da un entero no negativo b_t, se deberán dar b_t pasos de reloj al LFSR controlado antes de que éste dé su siguiente bit de salida u(t). Además, hay que dar un paso al LFSR de control para reiterar el proceso anterior. Si llamamos $s\{t\}$ a la salida del LFSR controlado, si no se aplica sobre éste la técnica de control de reloj, su salida $u\{t\}$ cuando sí se aplique esta será:

$$u(t) = s\left(\sum_{k=0}^{t} b_k\right) \qquad \text{para } t \geq 0$$

si la salida inicial es $u(0) = s(b_0)$. Además, se debe tener en cuenta que el LFSR de control tendrá una salida periódica de periodo M, con lo que $b_{t+M} = b_t$ para todo $t \geq 0$. Entonces se cumplirá que:

$$u(i + jM) = s(jp + c(i)) \qquad 0 \leq i < M \qquad j \geq 0$$

donde:

$$c(t) = \sum_{k=0}^{t} b_k \quad (1) \qquad \text{y} \qquad p = \sum_{k=0}^{M-1} b_k \quad (2)$$

ya que $c(t + M) = c(t) + p$ para todo $t \geq 0$. Además, se debe imponer la condición de que p y N sean mutuamente primos entre sí, y sea N el orden del polinomio de realimentación f(x).

Un caso especial de gran importacia es cuando b_t toma solamente los valores 0 y 1, dependiendo de cuál sea el bit de salida del LFSR de control. El generador resultante se llama de marcha y espera (*stop & go*), y se verá con más detalle en el apartado 4.2.4. Otro caso es cuando el registro de control no es único, sino que es una cascada de generadores (es el caso del generador de Gollmann que se estudiará en el apartado 4.2.4 B).

A continuación se verán algunos generadores de secuencias pseudoaleatorias basados en esta técnica de control de reloj.

4.2.1 Generadores de control de reloj con modulación de fase

En una descripción de control de reloj como la del apartado anterior, los valores de b_t deberían ser bastante pequeños, ya que si no el LFSR controlado necesitará funcionar a una velocidad mucho mayor que la tasa a la que produciría bits de salida. Para solucionar esto, se puede modular en fase la salida $u(t)$ para producir una salida $u'(t) = u(t + \lambda_t)$ de forma que se obtenga el mismo efecto que cuando se tienen b_t grandes, donde $\{\lambda_t\}$ es una secuencia de enteros de periodo M [CHA88]. A partir de aquí, se pueden obtener unas ecuaciones similares a las del apartado anterior. Para este caso de modulación de fase se cumplirá que $u'(t) = s(c'(t))$, donde $c'(t) = c(t + \lambda t)$ y además $c'(t)$ cumplirá la condición $c'(t + M) = c'(t) + p$.

Al igual que en el apartado anterior, también se puede definir una b'_t como $b'_t = c'(t) - c'(t-1)$ y que satisface la condición $b'_{t+M} = b'_t$.

El LFSR de control y la secuencia $\{\lambda_t\}$ deben tener periodos diferentes, por ejemplo M' y M'' respectivamente. Entonces podemos definir el periodo de control global como el menor múltiplo común entre M' y M'' y el valor p visto anteriormente vendrá dado en este caso por una suma sobre M'/M'' periodos del LFSR controlado, y será, por tanto, un múltiplo de $M'M'$. Por tanto, para mantener la condición de que p sea mutuamente primo con N (donde N es el orden del polinomio irreducible del LFSR controlado), se debe cumplir que M/M' sea mutuamente primo con N. Veamos algunos generadores de secuencias pseudoaleatorias que se basan en esta técnica.

A) El generador mezclador de modulación de fase de MacLaren-Marsaglia

Como ejemplo de este tipo de generadores se encuentra el llamado mezclador de MacLaren-Marsaglia [MAC65], cuya estructura muestra la figura 4.11, y que usa un generador pseudoaleatorio auxiliar para producir una secuencia periódica de enteros $\{x_t\}$ en el rango, por ejemplo, de 0 a K - 1.

Estos enteros se usan para seleccionar una dirección en una memoria de acceso aleatorio (RAM) de K direcciones, cada una de ellas almacena un bit. El bit que contiene la posición determinada por x_t se lee para producir el nuevo bit de salida $u'(t)$, y en su lugar se escribe el valor que contiene $u(t)$.

Los valores de K elevados no son recomendables, no sólo por el hecho de que se necesitará una RAM de elevada capacidad (y por tanto elevado coste), sino también por el hecho de que este generador necesita una inicialización que concluirá cuando todas las posiciones de la RAM se hayan llenado.

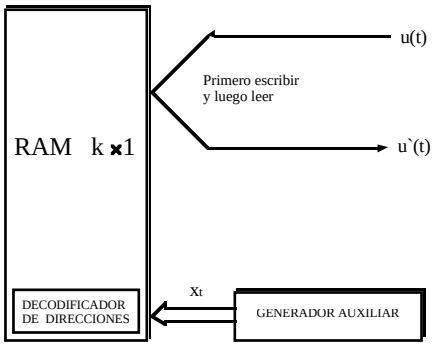


Fig. 4.11 Generador de MacLaren-Marsaglia

Un ejemplo práctico de este generador consistiría en usar este sistema para mezclar la salida de un LFSR de longitud n con un polinomio primitivo de realimentación de orden $N = 2^n-1$ controlado mediante una técnica de control de reloj. Esta, a su vez, puede consistir en emplear otro LFSR de longitud n y periodo $M' = N$ cuya salida ataque a la entrada de reloj del anterior. Si elegimos el periodo del mezclador M'' de forma que sea primo y además mutuamente primo con N, el periodo global del control será $M = N \times M''$, y la complejidad lineal de la secuencia de salida estará acotada inferiormente por $N \times n$.

Si bien el empleo de esta técnica no mejora la complejidad final respecto al LFSR con entrada de reloj contralada por otro LFSR, sí mejora, sin embargo, las propiedades estadísticas de la secuencia pseudoaleatoria resultante.

B) Generador de producto escalar con el vector estado

Este es otro tipo de generador que realiza una modulación de fase de forma más potente que el anterior, y se basa en el producto escalar binario [LID86]. El producto escalar binario de dos vectores es el producto escalar reducido módulo 2. En la figura 4.12 se muestra cómo implementar este generador. En ella, hay un LFSR de control de periodo M' cuya salida ataca a la entrada de reloj de otro LFSR de n celdas con polinomio de realimentación irreducible f(x). Este LFSR controlado producirá una secuencia de salida $S_t = S(c(t)) = (s_0(t), s_1(t),..., s_{n-1}(t))$ en el instante t según las pautas que se han visto anteriormente ($c(t)$ está definida en la ecuación 1 del apartado 4.2), que se denominará el vector de estado. En esta misma estructura existe un generador pseudoaleatorio auxiliar que elige direcciones en una memoria de sólo lectura (ROM) que contiene n vectores. La salida de este subsistema constituído por el generador auxiliar y la ROM la denotamos por $U_t = \{U_t\}$, y tendrá un cierto periodo M''. La salida $W(t)$ de toda esta estructura en el instante t será el producto escalar de las secuencias S_t y U_t. El periodo de control global será igual al menor múltiplo común entre M' y M'', por lo que será además el periodo de la secuencia de salida $W(t)$. Tendremos, pues, que $W(i + jM) = U_i \cdot S(c(i) + jp)$ con $0 \le i < M$ y $j \ge 0$, donde el valor de p viene dado en la ecuación 2 del apartado 4.2. Para un valor i fijo, $\{W(i + jM)\}$ con j = 0, 1, 2,... será una p-decimación de la secuencia producida por el LFSR, cuyo reloj es controlado. $W(t)$ consistirá, pues, en M secuencias mezcladas.

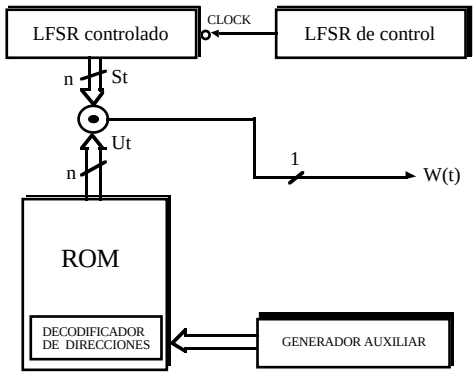


Fig. 4.12 Generador de producto escalar con el vector estado

La ventaja de este generador con respecto al del anterior apartado es que el rango de valores de $\lambda(U_i)$ puede llegar a ser del orden de 2^n-1.

Una variación de este generador podría obtenerse al sustituir el LFSR auxiliar y la memoria ROM por un LFSR de longitud mayor que *n*, del cual *n* bits se tomarían para formar la secuencia de vectores $\{U_t\}$. También se podría utilizar otro tipo de función combinadora no lineal diferente al producto escalar.

C) Generador en cascada

Se podrían conseguir mayores complejidades lineales que en las estructuras anteriores si el periodo *M* del LFSR que realiza el control del reloj fuera una potencia (N^r) del periodo *N* del LFSR cuyo reloj es controlado.

En este caso, el periodo de la secuencia de salida del conjunto sería $P = N^{r+1}$. Esto sugiere que podría ser interesante emplear estructuras en cascada de LFSRs, todas ellas de periodo *N*, y cuyo reloj esté controlado por el anterior.

Gollmann [GOL84] considera una estructura de este tipo que se describe en el apartado 4.2.5.B, en la que el control de reloj entre los sucesivos LFSR de la cascada se realiza mediante un solo bit. En este apartado se describe, sin embargo, un generador en el que el control entre las diferentes etapas de la cascada se realiza mediante varios bits en paralelo.

Para ello se construye una cascada de *L* etapas (excepto la primera y la última) como la de la figura 4.13. La entrada a cada una de estas etapas (a_t) consistirá en una secuencia de enteros de *k* bits. Uno de estos *k* bits se debe retrasar un paso, y se usará para elegir si al LFSR de *n* celdas y polinomio de realimentación primitivo se le deben dar uno o dos pulsos de reloj. Esto quiere decir que el LFSR se desplazará después de que se haya usado su salida b_t. Esta salida consistirá en *k* bits correspondientes a *k* celdas del LFSR ($k \leq n$), y éstos se deben sumar (módulo dos) bit a bit con los *k* bits de a_t para dar el vector suma c_t de *k* bits, que será además la entrada a una caja-*S* invertible que realizará una permutación de los enteros entre 0 a 2^k-1. Invertible significa que la permutación que debe realizar se hace mediante la transformación correspondiente, uno a uno, entre los enteros del rango considerado.

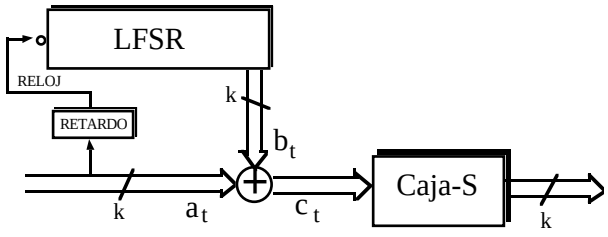


Fig. 4.13 Generador en cascada

La primera de las *L* etapas de este generador será un LFSR de *n* celdas con pulsos de reloj a intervalos regulares (por tanto no controlado) seguido de una caja-*S* invertible. La última etapa será como la de la figura 4.13, pero omitiendo la caja-*S*.

El periodo *P* de las *k* secuencias producidas por este generador será $P = N^L$, y todas ellas tendrán una complejidad lineal de al menos N^{L-1}. Si no se omite la caja-*S* de la última de las *L* etapas, la complejidad lineal de las *k* secuencias de salida aún puede incrementarse más.

Las cajas-*S* pueden ser diferentes para las distintas etapas, así como los polinomios primitivos de realimentación, pero siempre que el número de celdas de los LFSRs sea *n*.

D) El generador multiplicador de tasa binaria (BRM)

Este generador de secuencias pseudoaleatorias, propuesto por W.G. Chambers y S.M. Jennings en 1984 [CHA84] presenta la estructura de la figura 4.14.

Sea el LFSR-1 un registro de desplazamiento de *m* etapas que genera una secuencia de periodo $M = 2^m-1$ y el LFSR-2 un registro de desplazamiento de *n* etapas que genera una secuencia de periodo $N = 2^n-1$.

El generador BRM funciona de la siguiente forma: En el instante *t*, se da un pulso de reloj a ambos registros de desplazamiento. Entonces se hace que el LFSR-2 reciba a_t pulsos de reloj más, si a_t es el número que se obtiene de examinar el contenido de *k* celdas del LFSR-1. Para ello se debe asumir que $k < m$, de forma que $0 \le a_t \le 2^k-1$. La salida *z(t)* del sistema será la salida del LFSR-2 después de que ambos registros de deplazamiento hayan hecho un desplazamiento, y se haya desplazado al LFSR-2 a_t veces más. Si llamamos $\{s(t)\}$ a la secuencia generada por el LFSR-2, y $\{z(t)\}$ a la secuencia de salida del sistema, podemos representar al sistema para $t = 0, 1, 2,...$ como:

$$z(t) = s\left(t + \sum_{i=0}^{t} a_i\right)$$

puesto que siempre se cumple que $a_{t+M} = a_t$ para todo instante de tiempo *t*, para $0 \le i \le M-1$ y para todo *j* tendremos que $z(i + jM) = s(jp + d_i)$ donde:

$$d_i = i + \sum_{l=0}^{i} a_l \qquad \text{y} \qquad p = M + \sum_{l=0}^{M-1} a_l$$

Durante un ciclo completo de *M* pulsos de reloj producidos por el LFSR-1, se producirán cada una de las *m*-tuplas binaria distintas de cero como estados del LFSR-1.

Esto implica que cada *k*-tupla (excepto la de *k* ceros) entrará en el bloque BRM 2^{m-k} veces, por tanto:

$$\sum_{l=0}^{M-1} a_i = 2^{m-k} \sum_{i=1}^{2^k-1} i = 2^{m-1}(2^k - 1)$$

y

$$p = M + 2^{m-1}(2^k - 1)$$

Si se cumplen las condiciones de que todo factor primo de *M* divida a *N*, y que *p* sea primo con *N*, la complejidad lineal de la secuencia de salida $\{z(t)\}$ será $M \times n$. En la práctica deberemos elegir $M = N$. En este caso la condición de que *p* y *N* sean primos entre sí, se cumplirá si, y sólo si, *k* es primo relativo con *m*. Por tanto, si *m* es un número primo y $m = n$, entonces cualquier valor de *k* dará a la salida una secuencia de complejidad lineal $M \times n$ y periodo $M \times N$.

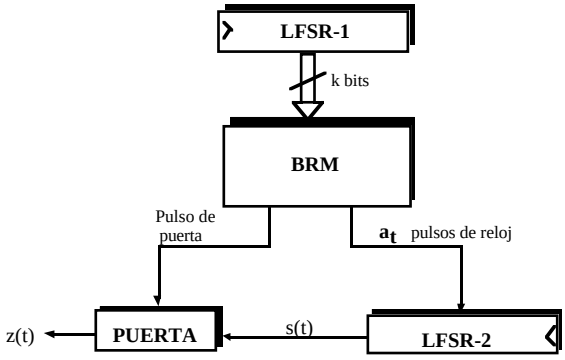


Fig. 4.14 Generador multiplicador de tasa binaria

Veamos un ejemplo. Supongamos que ambos registros de desplazamiento tienen 127 celdas y tienen polinomio de realimentación primitivo. Con estos valores, el periodo *M* será 2^{127}-1 y la complejidad lineal $127(2^{127}\text{-}1) = 2 \times 16 \times 10^{40}$, que es un valor considerablemente superior que el valor 127 que obtendríamos con un único registro de desplazamiento del mismo tipo. El periodo del generador BRM en este caso será $(2^{127}\text{-}1)^2 = 2{,}895 \times 10^{76}$.

4.2.2 Generador de secuencias autodecimadas

Los generadores de secuencias autodecimadas se basan en producir el avance del reloj que controla al generador en función del valor del bit de la propia secuencia de salida. A este proceso se le denomina decimación del registro de desplazamiento. Vamos a ver dos generadores que se basan en esta

técnica. El primero de ellos (generador de Ruepple) realiza una decimación de la secuencia de salida por un valor constante *d* que consiste en tomar muestras de la secuencia de salida a una velocidad *d* veces menor que el cambio de estado interno del registro de desplazamiento con realimentación lineal. El segundo generador (generador de Chambers-Gollmann) realiza una decimación (*d*,*k*) que, a diferencia de la anterior no es constante, sino que se realiza por una función dependiente de los dígitos previos de la secuencia del LFSR generador por lo que se obtiene como consecuencia una secuenia (*d*,*k*)-decimada.

Si bien en el capítulo anterior ya se vieron las propiedades de la decimación de secuencias, veamos algunas consideraciones antes de pasar a ver los generadores basados en esta técnica.

Convencionalmente, se define la decimación de una secuencia $s(t)$ $t = 0, 1, 2,...$ por un valor constante *d* como la extracción de cada *d*-ésimo dígito de $s(t)$, y usualmente se denota por $s[d]$. Por el contrario, también es posible realizar decimaciones no constantes sobre la secuencia, de forma que dependa de porciones previas de ésta. Las secuencias producidas de esta forma se denominan [*d*,*k*]-autodecimadas. Veámoslo con un ejemplo:

EJEMPLO 4.1: Sea la siguiente porción de una *m*-secuencia $s(t)$= 1111100100110000101101010 0..., producida por un registro de desplazamiento con realimentación lineal. Una secuencia 2-autodecimada a partir de la secuencia dada será:

$$s[2]= 1110010011000...$$

mientras que una secuencia [1,2]-autodecimada será:

$$s[1,2]=11101010000110110....$$

Para realizar el estudio de las secuencias autodecimadas se supondrá que las secuencias originales son *m*-secuencias (secuencias binarias de máximo periodo producidas por registros de desplazamiento con realimentación lineal). Esto implicará que $0 < d,k < 2^L-1$, si *L* es el número de celdas del LFSR. Sin embargo, para eliminar la restricción anterior bastará que aquellos *d* ó *k* que sean mayores que 2^L-1 se reduzcan mediante la operación mod 2^L-1. Las secuencias [*d*,*k*]-autodecimadas son tan fáciles de implementar como las propias *m*-secuencias, y si los valores de *d* y *k* se eligen adecuadamente, presentan tan buenas propiedades de distribución (excelente *k*-distribución) como éstas, con la ventaja adicional que tienen una complejidad lineal muy elevada, de valor similar al periodo, por lo que parecen comportarse de una forma más aleatoria que las *m*-secuencias. Estos generadores pueden tener buena aplicación en modulaciones *spread spectrum* y criptografía (ambas aplicaciones se estudiarán en el capítulo 7). Sin embargo, se debe tener precaución, ya que si para realizar un cifrado se usa una sola secuencia [*d*,*k*]-autodecimada a partir de una *m*-secuencia y el par [*d*,*k*] se hace público, un posible atacante podría obtener el polinomio de realimentación y el estado inicial a partir de la secuencia de salida mediante la resolución de un sistema de ecuaciones lineales.

A) Generador de secuencias autodecimadas de Ruepple

La estructura de este generador de secuencias pseudoaleatorias, propuesto por R.A. Ruepple en 1987 [RUE88] es muy simple, tal como muestra la figura 4.15. Se basa en un único registro de

desplazamiento con realimentación lineal con polinomio de realimentación primitivo, y cuya secuencia de salida se usa para controlar el propio reloj que marca la producción de los bits, de forma que si a la salida aparece un 0, el reloj produce *d* pulsos, y si aparece un 1, el reloj produce *k* pulsos. El efecto que causa esto es que la *m*-secuencia que produce el LFSR es decimada para formar la secuencia de salida de acuerdo con una regla pseudoaleatoria que viene determinada por la propia *m*-secuencia. Este generador no es muy bueno para aplicaciones criptográficas.

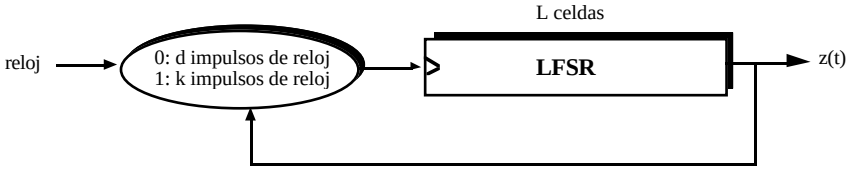


Fig. 4.15 Generador de secuencias autodecimadas de Ruepple

B) Generador de secuencias autodecimadas de Chambers y Gollmann

Otro generador basado en la autodecimación de secuencias es el propuesto por Chambers y Gollmann [CHA88] como una modificación del generador de secuencias autodecimadas de Ruepple, y cuyo esquema se muestra en la figura 4.16. El propósito final de este generador es disponer de un elemento básico a partir del cual construir estructuras mayores mediante cascada. Veamos cuál es su estructura y cuáles son sus propiedades: este generador está formado por un registro de desplazamiento con realimentación lineal con polinomio de realimentación primitivo cuyo reloj está controlado por el resultado de aplicar una función NOR sobre las *p* celdas de menor peso. Si el resultado de esta función es un 0, el registro de desplazamiento da un solo paso, y si es un 1, da dos pasos.

Con este generador se pretende conseguir que la secuencia de salida presente un periodo similar al máximo posible $T = 2^L-1$ para un registro de desplazamiento realimentado linealmente de L celdas y una complejidad lineal igual a T, o a $T-1$. Sin embargo, esto es muy delicado, puesto que a medida que aumenta p, las propiedades estadísticas de la secuencia autodecimada resultante se alejan más y más de las de la *m*-secuencia producida por el LFSR original. Hay, pues, que tener en cuenta ciertas restricciones entre el número de celdas (L) del LFSR, y el número de celdas que se emplean como entrada de la función NOR (p) para conseguir que la complejidad lineal de la secuencia de salida sea igual a su periodo, y ambos estén próximos al valor máximo obtenible 2^L-1. La restricción que debe cumplir el par (p, L) para obtener estas condiciones viene determinado por una relación respecto al periodo, expresada por:

$$T = (2^L - 1) - \frac{2(2^{L-p} - 1)}{3}$$

si (L-p) es par (condición necesaria para que el periodo T sea impar) y tanto el periodo T como la complejidad lineal son números 2-primos. Un número 2-primo es el que satisface que 2^i-1 no es

múltiplo de T para cualquier $i < T$-1. Una manera rápida de saber si T es 2-primo se basa en el hecho de probar los i, tales que valgan $(T\text{-}1)/T'$, si T' es cada uno de los los factores primos de T-1. La condición anterior se encuentra cuando se estudian cuántos ceros y unos se deben restar (en total $2\left(2^{L-p}-1\right)/3$) al periodo de la m-secuencia (2^L-1). Cuanto más parecidos son L y p, el periodo de salida del generador (T) y, por tanto, también su complejidad, tanto más se acercarán al máximo valor posible 2^L-1.

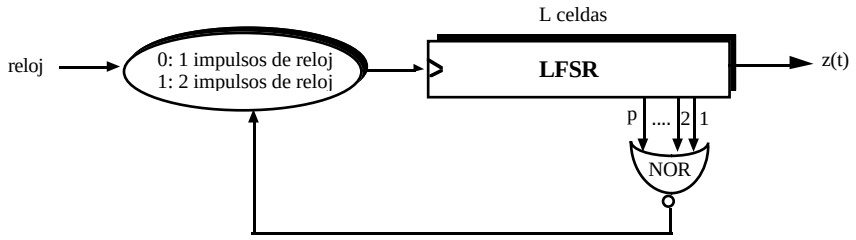


Fig. 4.16 Generador de Chambers-Gollmann

Para estudiar el comportamiento de este generador podemos hacer una comparación entre la secuencia producida por el generador (con la función NOR) y la m-secuencia original. Si para ello elegimos un LFSR de 5 celdas (L = 5) con polinomio de realimentación primitivo estas secuencias obtenidas son:

m-secuencia: 100<u>0</u>1101110101000<u>0</u>10<u>0</u>1<u>0</u>1110<u>0</u>1<u>1</u>111 (periodo = 31)

generador: 10011011101010000001100111 (periodo = 26)

Se puede ver que la secuencia obtenida por el generador se obtiene a partir de la m-secuencia pero eliminando los bits subrayados, ya que el generador se los ha saltado al dar dos pasos cuando ha encontrado la condición de p ceros consecutivos en la secuencia de salida.

Una cuestión importante a la hora de diseñar un generador de este tipo es que si el vector inicial coincide con uno de los vectores que se saltan pueden darse preámbulos en la secuencia antes de que se convierta en periódica. Estos preámbulos son siempre indeseables ya que reducen el periodo de la secuencia de salida en su misma longitud. Afortunadamente, se sabe que la máxima longitud que puede tener un preambulo es $\lfloor L - p + 1 \rfloor$ (donde $\lfloor x \rfloor$ indica la parte entera de x). Se puede estudiar cuántos estados iniciales hay globalmente buenos para evitar los preámbulos. Por ejemplo, para un LFSR con L = 5 y periodo T = 29, el número de polinomios primitivos de realimentación con periodo 31 es $\phi(2^L - 1)/L = 6$, y por lo tanto hay como mucho 6(31-29) = 12 estados que producirán preámbulos como mínimo en uno de esos polinomios de realimentación. Por tanto, se dispone de 31-12 = 19 estados iniciales globalmente buenos.

Para concluir el estudio de este generador, la tabla 4.2 muestra las combinaciones (L,p) para $L \leq 20$ que cumplen que el periodo de salida del generador (T) es un 2-primo. Se puede observar que,

aunque el límite teórico ofrece un rendimiento $T/2^L-1 = 66\%$, se garantizan rendimientos por encima del 80% ($4 < L \leq 70$), e incluso no es difícil encontrar rendimientos por encima del 99%. Por tanto, se puede decir que para valores $4 \leq L \leq 20$ se puede garantizar la obtención de secuencias con periodos y complejidades lineales del mismo valor y muy cercano el máximo teórico $T_{max} = 2^L-1$.

Tabla 4.2 Combinaciones (L,p) para $L \leq 20$

L	p	*T*	*$T/2^L-1$*
3	1	5	0,625
4	2	13	0,812
5	3	29	0,906
6	2	53	0,828
6	4	61	0,953
9	7	509	0,994
10	2	853	0,833
11	3	1877	0,916
12	2	3413	0,833
12	10	4093	0,999
14	12	16381	0,999
16	6	64853	0,989
17	5	128341	0,979
18	2	218453	0,833
18	10	261973	0,999
18	14	262133	0,999
19	7	521557	0,994
20	18	1048573	0,999

4.2.3 El generador de marcha y parada (*Stop & Go*)

Este tipo de generadores de secuencias pseudoaleatorias también se basa en controlar los impulsos de reloj que actúan sobre uno o varios generadores de *m*-secuencias, usando para ello una secuencia controladora de pasos. Veamos algunos generadores que se basan en esta técnica.

A) Generador de marcha y espera de Beth-Piper

Este generador de secuencias pseudoaleatorias [BET85] se basa en la aplicación de las técnicas de control de reloj descritas en el capítulo 3, y su estructura se puede ver en la figura 4.17. Su funcionamiento es el siguiente: El reloj que actúa sobre el registro de desplazamiento con realimentación lineal LFSR-2 está controlado por la salida del LFSR-1, de forma que el LFSR-2

cambiará de estado en el instante *t*, si y sólo si, la salida del LFSR-1 en el instante anterior es un 1 ($x_1(t\text{-}1) = 1$). Además, la salida del LFSR-2 se combina con la salida de un tercer LFSR (LFSR-3) y funciona de forma normal. Esta combinanción se realiza mediante una suma, bit a bit, de la salida de los LFSR 2 y 3, con lo que se obtiene la salida global del generador *z*(t). Si se imponen las restricciones adecuadas sobre los grados de los tres LFSR (n_1, n_2 y n_3 para los LFSR-1, LFSR-2 y LFSR-3 respectivamente) y sus polinomios son primitivos, $z(t)$ tendrá periodo P y complejidad Λ:

$$P = (2^{n_1} - 1)(2^{n_2} - 1)(2^{n_3} - 1) \qquad \Lambda(z) = (2^{n_1} - 1)n_2 + n_3$$

Sin embargo, aunque la complejidad lineal de la secuencia de salida del generador es elevadísima, este generador presenta ciertas dependencias estadísticas entre las secuencias que producen los diversos LFSR´s, que hace que su uso no sea adecuado en criptografía, ya que esta debilidad se puede emplear para criptoanalizar el sistema. Veamos esto: Sea $x_2`(t)$ la secuencia de salida del LFSR-2 en el instante *t* si suponemos a éste en libre funcionamiento (fuera del conjunto global y, por tanto, con pulsos de reloj normales). Entonces tendremos la siguiente probabilidad de coincidencia:

$$p = Prob[z(t) \oplus z(t+1) = x_3(t) \oplus x_3(t+1)] =$$
$$= Prob(x_1(t) = 0) + [Prob(x_1(t) = 1) \cdot Prob(x_2'(t) = x_2(t+1))] = \frac{1}{2} + \frac{1}{4}$$

El que esta probabilidad sea superior a 1/2 implica que si los polinomios de realimentación de los LFSR-1 y LFSR-3 son conocidos por un posible atacante, éste puede aplicar un ataque basado en el método del síndrome lineal [ZEN89], que permita primero recuperar la secuencia $x_3(t)$ a partir de la secuencia de salida $z(t)$, y después la secuencia $x_1(t)$ a partir de la $x_2(t)$. Ello se podría hacer incluso bajo la suposición de que el polinomio de realimentación del LFSR-2 no fuera conocido.

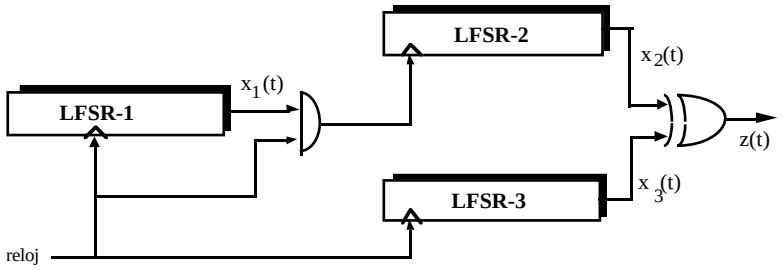


Fig. 4.17 Generador de Beth-Piper

B) Generador de marcha y espera en cascada de Gollmann

Este generador de secuencias pseudoaleatorias [GOL85] consiste en una variación del generador de Beth-Piper visto en el apartado anterior, y su esquema se puede ver en la figura 4.18.

Consiste en una serie de *N* registros de desplazamiento con realimentación lineal y polinomio de realimentación primitivo todos ellos del mismo grado *n*. El funcionamiento de este generador es el siguiente: El reloj que marca el funcionamiento del LFSR *i*-ésimo está controlado por todos los LFSR *j*-ésimos (con $j < i$ de forma conjunta). Este control se realiza de la misma manera que se vio en el generador de marcha y espera de Beth-Piper.

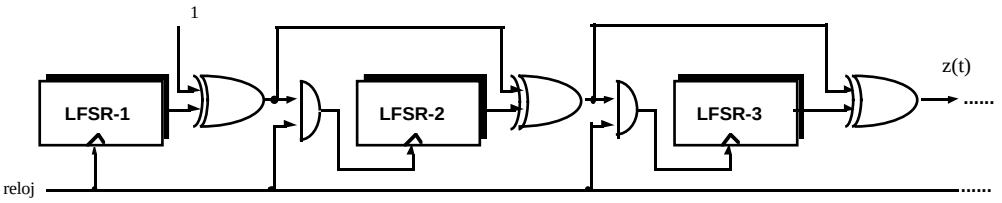


Fig. 4.18 Generador de marcha y espera en cascada de Golmann

Mediante esta técnica se consigue que la secuencia de salida $z(t)$ tenga el siguiente periodo:

$$T = (2^n - 1)^N$$

y una complejidad lineal muy elevada, acotada inferiormente mediante la expresión:

$$\Lambda(z) \geq n(2^n - 1)^{N-1}$$

C) Generador de marcha y espera de control bilateral

Este generador de marcha y espera [ZEN90] se basa en el hecho de que si una secuencia binaria *s* tiene un periodo *T* que es un primo impar, su complejidad lineal $\Lambda(s)$ estará acotada inferiormente por el orden del número 2 módulo *T* ($\mathrm{Ord}_T(2)$), es decir:

$$\Lambda(s) \geq Ord_T(2)$$

Cuando se use este generador en aplicaciones como la criptografía, el hecho de elegir una secuencia de periodo primo es deseable, debido a que se puede someter a diversas transformaciones sin que se modifique la cota inferior de su complejidad lineal, suponiendo que la transformación utilizada no nos conduzca a una secuencia constante. Sin embargo, si el periodo $T = 2^n\text{-}1$ resultara ser un primo de Mersenne, entonces $\mathrm{Ord}_T(2) = n$, y ello implica que el periodo *T* que elijamos debe ser un primo superior a $2^n\text{-}1$. De este modo podremos construir secuencias con periodos de la forma $T = q\ 2^n\text{-}1$, si *q* es un número impar mayor o igual que 3, y hay una estructura formada por dos registros de desplazamiento con realimentación lineal (LFSR) ambos de *n* celdas [ZEN90].

Un generador de estas características se muestra en la figura 4.19. Con esta estructura se puede obtener un periodo $T = 5 \cdot 2^{n-2} - 1$. Para ello, ambos LFSR de *n* celdas deben inicializarse con

estados iniciales distintos de cero. El control de los pasos en este generador se realiza de la siguiente forma:

(a) Si $(x(t+n-1),x(t+n-2)) = (0,1)$ se bloquea el pulso de reloj del LFSR-2.

(b) Si $(y(t+n-1),y(t+n-2)) = (0,1)$ pero $(x(t+n-1),x(t+n-2)) \neq (0,1)$, se bloquea el pulso de reloj del LFSR-1.

El diagrama de estados de este generador consta de $3 \cdot 2^{n-2} - 1$ ciclos ramificados de longitud $5 \cdot 2^{n-2} - 1$, cada uno de los cuales presenta un cierto número de ramas de longitud uno. Los valores posibles de n que hacen que $5 \cdot 2^{n-2} - 1$ sea primo se pueden determinar usando el criterio de Lucas-Lehmer o bien mediante un algoritmo que se puede encontrar en [ZEN90]. En este generador, la complejidad de la secuencia de salida es del orden de magnitud del periodo.

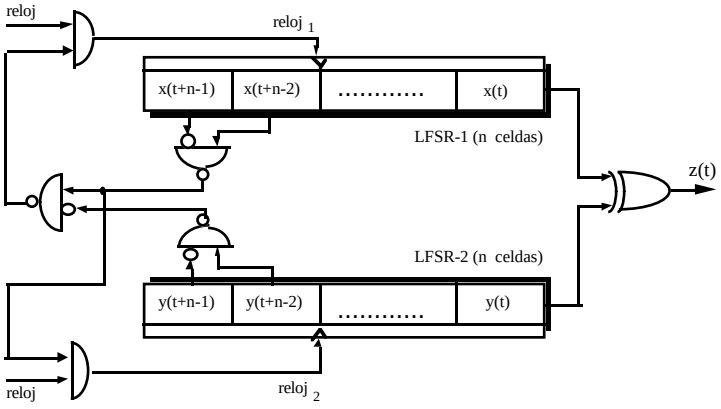


Fig. 4.19 Generador de marcha y espera de control bilateral

4.2.4 El generador de pasos alternados (ASG)

Este generador de secuencias pseudoaleatorias, propuesto por Günter en 1988 [GÜN88], comparte las ventajas de los generadores de marcha y espera vistos anteriormente, pero evita algunas de sus debilidades. Su estructura está formada por tres subgeneradores basados en registros de desplazamiento con realimentación lineal y su funcionamiento consiste en que la salida de uno de los LFSR (el LFSR-3 en la figura 4.20) controla el reloj que ataca a los otros dos registros de desplazamiento (LFSR-1 y LFSR-2). Este control se realiza de la siguiente forma: cuando el bit de salida del LFSR-3 es un 1, se le da un impulso de reloj al LFSR-1, y cuando es un 0, se da el pulso al LFSR-2. Si $x_1(t)$, $x_2(t)$ y $x_3(t)$ son las secuencias que producen los registros de desplazamiento LFSR-

1, LFSR-2 y LFSR-3 respectivamente, cuando se encuentran separados del generador (en funcionamiento independiente), la salida del generador ASG, $z(t)$, se obtiene sumando bit a bit módulo 2 las salidas de los LFSR-1 y el LFSR-2 ($z(t)= x_1(t) \oplus x_2(t)$).

En la práctica, los subgeneradores LFSR-1, LFSR-2 y LFSR-3 son generadores de *m*-secuencias (registros de desplazamiento con realimentación lineal con polinomio de realimentación primitivo). Sin embargo, en el artículo original de Günter el subgenerador 3 estaba constituído por un generador de secuencias de De Bruijn, debido a que ello simplificaba mucho su estudio teórico. Veamos algunos de los resultados obtenidos para este generador si el subgenerador 3 es un generador de secuencias de De Bruijn. Sin embargo, otras pruebas realizadas demuestran que los resultados usando un LFSR para el subgenerador 3 son muy similares.

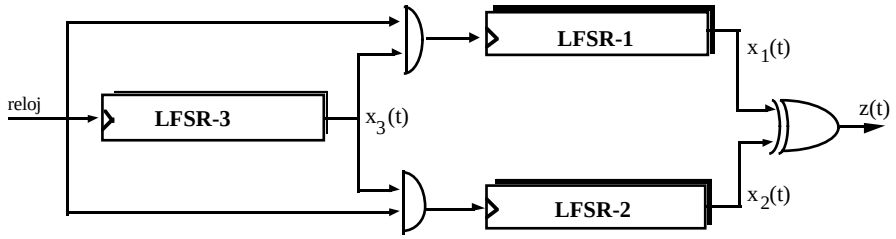


Fig. 4.20 Generador de pasos alternados ASG

Se estudiarán en primer lugar las características de periodo y complejidad lineal. Para facilitar la demostración se recurre a los siguientes teoremas:

TEOREMA 1: Si se efectúan las siguientes suposiciones:

(a) $x_3(t)$ es una secuencia de De Bruijn de periodo 2^k (si k es su número de celdas).
(b) Los polinomios de realimentación del LFSR-1 y del LFSR-2 son irreducibles y diferentes y tienen grados n_1 y n_2, y periodos T_1 y T_2 respectivamente.
(c) Los periodos T_1 y T_2 son mayores que 1 y primos entre sí: T_1, T_2 >1 y mcd(T_1,T_2) = 1.

Bajo estas suposiciones, el periodo de la secuencia de salida $z(t)$ del generador es:

$$T = 2^k \cdot T_1 \cdot T_2$$

y la complejidad lineal está acotada por la expresión:

$$(n_1 + n_2)2^{k-1} < \Lambda \leq (n_1 + n_2)2^k$$

TEOREMA 2**:** Si se considera la frecuencia de aparición de las subsecuencias cortas se pueden efectuar las siguientes suposiciones:

(a) $x_3(t)$ es una secuencia de de Bruijn de periodo 2^k (si k es su número de celdas).
(b) $x_1(t)$ y $x_2(t)$ son m-secuencias de periodos $T_1 = 2^{n_1} - 1$ y $T_2 = 2^{n_2} - 1$ respectivamente.
(c) Los periodos T_1 y T_2 son mútuamente primos entre sí (mcd(T_1,T_2) = 1.

Bajo estas suposiciones, la frecuencia de cualquier subsecuencia de longitud $w \leq min\{n_1, n_2\}$ es 2^{-w}, con un error del orden de $O\left(1/2^{n_1-w}\right) + O\left(1/2^{n_2-w}\right)$. La desviación de esta distribución respecto a la ideal es similar a la que presenta una m-secuencia. Este generador permite producir secuencias pseudoaleatorias de forma simple y eficiente, además de ser rápido.

Puede ser aconsejable formar una estructura en cascada utilizando este generador. Para ello, se sustituye cada LFSR por el generador de pasos alternados ASG. El generador así construído presentará elevado periodo, elevada complejidad lineal y buenas propiedades estadísticas en las secuencias obtenidas.

4.2.5 El generador multivelocidad de Massey-Ruepple

La estructura de este generador de secuencias pseudoaleatorias [MAS84] consiste en dos registros de desplazamiento con realimentación lineal (de grados m y n para el LFSR-1 y el LFSR-2 respectivamente, con $n \geq m$) con relojes funcionando a distinta velocidad, tal como muestra la figura 4.21. El reloj del LFSR-2 debe ser $d \geq 2$ veces más rápido que el del LFSR-1, con lo que la señal de salida del generador $z(t)$ vendrá dada por la expresión:

$$z(t) = \sum_{i=0}^{l-1} x(t+i)y(dt+i)$$

donde $x(t)$ e $y(t)$ son las secuencias producidas por el LFSR-1 y el LFSR-2 respectivamente.

Si los polinomios de realimentación utilizados por los registros de desplazamiento son primitivos, sus grados son mutuamente primos entre sí (mcd(m,n) = 1), y se cumple que el factor de velocidad d es mutuamente primo con el periodo del LFSR-2 (mcd(d,2^n-1) = 1), la secuencia de salida de este generador tendrá un periodo:

$$T = (2^n - 1)(2^m - 1)$$

y una complejidad lineal $\Lambda(z) = m \cdot n$. Además, la secuencia de salida producida por este generador presenta excelentes propiedades estadísticas.

Veamos cuál será la debilidad de este generador si se emplea en criptografía para realizar un sistema de cifrado en flujo. En este generador, el factor de velocidad d es variable y se puede emplear como parte de la clave secreta. La bilinearidad implicada en la generación de la secuencia de salida $z(t)$, fijado un valor del factor de velocidad, hace que el criptosistema se pueda romper por un ataque basado en el test de consistencia lineal [ZEN89]. Mediante este test, que se verá más detalladamente en el capítulo 7, si el atacante conoce los polinomios de realimentación de los registros de desplazamiento puede determinar el factor de velocidad d y los estados iniciales de los registros de desplazamiento (la

clave), aplicando tan sólo $(d_{max} - 1)(2^m - 1)$ tests de consistencia sobre una porción de la secuencia de salida de longitud $N \geq m + n + log_2 d_{max}$.

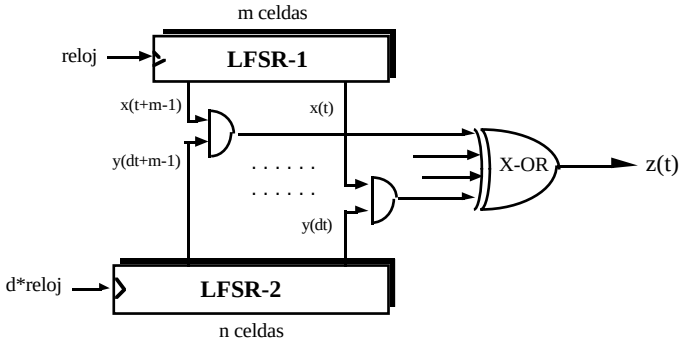


Fig. 4.21 Generador multivelocidad de Massey-Ruepple

4.2.6 El generador de producto interior

Consideremos dos registros de desplazamiento con realimentación lineal de L_1 y L_2 celdas respectivamente, cuyos relojes están atacados a velocidades diferentes (d_1 y d_2 respectivamente). La secuencia de salida $z(t)$ de este generador se obtendrá mediante el siguiente procedimiento:

FOR i = 1, 2,..... DO

Paso 1: Desplazar el $LFSR_1$ d_1 veces

Paso 2**:** Desplazar el $LFSR_2$ d_2 veces

Paso 3: Calcular el producto de los estados de los 2 generadores: $z_i = \sum_{i=1}^{min\{L_1, L_2\}} x_{1i} \cdot x_{2i}$

END FOR

donde las x_{ki}, con $k = 1, 2,..$ y $1 \leq i \leq min\{L_1, L_2\}$, son los contenidos de las celdas de ambos LFSR.

El periodo P de la secuencia de salida $z(t)$ estará acotado inferiormente por la expresión:

$$P(z) \geq P_1 P_2 / (q - 1)$$

y tendrá una complejidad lineal $\Lambda(z) = L_1 L_2$, suponiendo que los polinomios de realimentación de ambos registros de desplazamiento sean irreducibles, que los números de celdas sean mutuamente

primos entre sí (mcd(L_1,L_2) = 1), y además que mcd(d_1,P_1) = mcd(d_2,P_2) = 1, donde P_1 y P_2 son los periodos de los dos LFSR respectivamente.

El número de ceros en un periodo de la secuencia de salida será $(2^{L_1-1}-1)(2^{L_2}-1)$, y la diferencia entre el número de unos y ceros relativo a la longitud del periodo será $1/(2^{L_1}-1)$, considerando que los polinomios de realimentación sean primitivos y $L_1 > L_2$.

4.3 Generadores basados en otras técnicas

4.3.1 El generador de Windmill

El generador de Windmill [SMEET] presenta ventajas para la generación de secuencias pseudoaleatorias en modo bloque, es decir, para producir bloques de símbolos pseudoaleatorios. Además, presenta un paralelismo estructural que lo hace muy útil para realizaciones en VLSI (integración a muy alta escala). Un generador de Windmill consiste en una conexión en cascada cíclica de v ($v \leq 1$) registros de desplazamiento con realimentación lineal, tal como se muestra en la figura 4.22. Con cada LFSR se asocia un polinomio de realimentación $\alpha(t)$ y una red de realimentación lineal hacia el LFSR siguiente $\beta(t^{-1})$. A este conjunto se le llama un vano de Windmill. Según esta definición, el k-ésimo vano viene, pues, representado por el polinomio de realimentación hacia atrás $\alpha(t) = 1 - \sum_{j=1}^{m} \alpha_j t^j$ (ya que usa m celdas), y el polinomio de realimentación hacia adelante $\gamma_k(t) = t^{\lambda(k)}\beta(t^{-1})$ donde $\beta(t^{-1}) = \sum_{j=0}^{n} \beta_j t^{-j}$ (ya que usa n celdas), y $\lambda(k)$ representa el número total de celdas que tiene el registro de desplazamiento del vano, de forma que $\lambda(k) > \max(n,m)$. Cada vano tiene idénticas $\alpha(t)$ y $\beta(t^{-1})$. La primera celda del LFSR de cada vano se toma para formar un bloque de v bits y, para obtener la salida, éstos se deben combinar. La forma en que los v símbolos se combinan para producir la v-tupla final está controlada por una permutación σ. Así pues, la secuencia z de salida es:

$$\mathbf{z} = x_0^{\sigma^{-1}(0)},\ldots,x_0^{\sigma^{-1}(v-1)},x_1^{\sigma^{-1}(0)},\ldots,x_1^{\sigma^{-1}(v-1)},x_n^{\sigma^{-1}(0)},\ldots,x_n^{\sigma^{-1}(v-1)}$$

El generador completo se puede representar por $[\alpha(t),\beta(t^{-1}),\underline{\lambda},v,\sigma]$ donde $\underline{\lambda} = (\lambda(0),\ldots,\lambda(v-1))$ representa el vector que contiene los números de celdas de los v LFSRs del generador. Para cada vano k (k = 0, 1, ..., v-1), y para un $i \in N$ se tendrá un estado inicial $x_0^k, x_{-1}^k,\ldots,x_{-\lambda(k)+1}^k$ y la ecuación de recurrencia:

$$x_{i+1}^k = \sum_{j=1}^{m} \alpha_j x_{i+1-j}^k + \sum_{j=0}^{n} \beta_j x_{i+j-\lambda(k-1)+1}^{k-1}$$

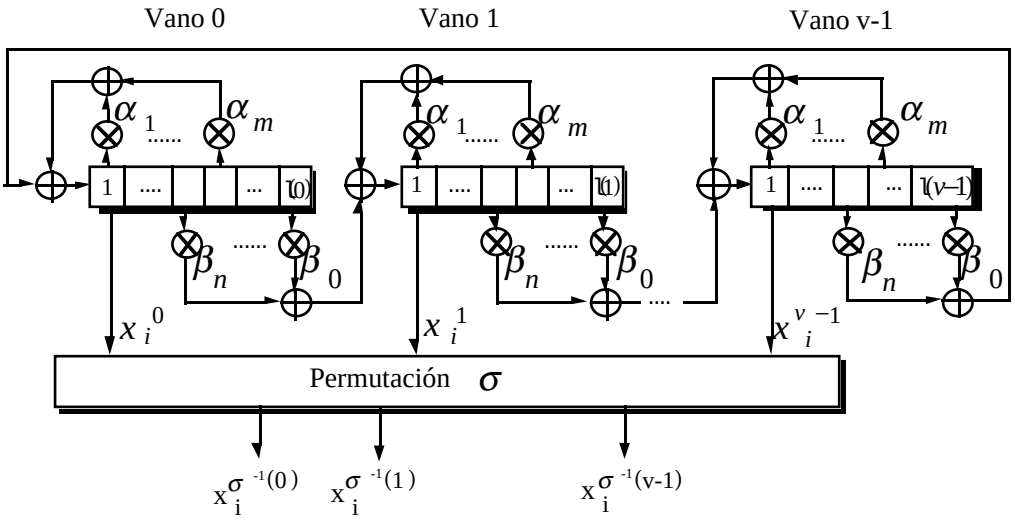


Fig. 4.22 Generador de Windmill

Podemos definir un polinomio de Windmill como $f(t) = \alpha(t^{\nu}) - \beta(t^{-\nu})t^{L}$, donde L y ν sean enteros tal que $1 \leq \nu < L$ y además sean mutuamente primos entre sí. Además, sean $\alpha(t)$ y $\beta(t^{-1})$ dos polinomios sobre GF(q) de grado positivo $m < L/\nu$ y $n < L/\nu$ respectivamente, de forma que $\alpha(0) = 1$ y $\beta(0) \neq 0$.

Con estas condiciones, si f(t) es un polinomio de realimentación primitivo sobre GF(q), existirá una permutación σ de los números 0, 1,..., ν-1, y un conjunto de longitudes de los registros de desplazamiento $\underline{l}$ definidas como:

$$\sigma(k) = Lk + c \pmod{\nu}$$
$$l(k) = (\sigma(k) - \sigma(k+1) + L)/\nu$$

donde k = 0, 1,..., ν-1, y c es un número fijo, la estructura $[\alpha(t), \beta(t^{-1}), \underline{l}, \nu, \sigma]$ generará una m-secuencia z de la siguiente forma:

$$z(t) = \frac{\sum_{k=0}^{\nu-1} t^{\sigma(k)} p_k(t^{\nu})}{f(t)}$$

donde p_k vendrá definida como:

$$p_k = p_k(t) = x_0^k + \sum_{j=1}^{m}\sum_{i=1}^{j-1} \alpha_j x_{i-j}^k t^i + \sum_{j=0}^{n}\sum_{i=-l(k-1)+1}^{-j-1} \beta_j x_{i+j}^{k-1} t^{i+l(k-1)}$$

Por tanto, si el polinomio f(t) es primitivo, la secuencia $z(t)$ producida será una m-secuencia. Además, si el grado de $\beta(t^{-1})$ es igual a $\lfloor L/v \rfloor$, entonces al menos uno de los vanos tendrá su entrada conectada a la salida del vano mediante la realimentación hacia adelante. Esta conexión debe ser considerada cuidadosamente, ya que puede dar problemas de temporización en aplicaciones prácticas. A los polinomios de Windmill que no producen esta clase de conexiones, y evitan así problemas de temporización, se les llama polinomios propios de Windmill, y cumplen la restricción adicional $v(grad\beta(t^{-1}) + 1) \leq L$. Se puede hacer además, y sin pérdida de generalidad, que el parámetro c valga 0, de forma que los valores de $\gamma(k)$ y $\sigma(k)$ dependan sólo de L y de v. Veamos ahora el número de polimomios de Windmill f(t), tanto irreducibles (N_i) como primitivos (N_p), de que podemos disponer. Para ello, los consideramos como un subconjunto de los polinomios de grado L con f(0)=1. Bajo esta condición, el número de polinomios de Windmill irreducibles que satisfacen que f(0)=1 será de:

$$N_i = \frac{2^{1+2\lfloor L/v \rfloor}}{L}$$

y el número de polinomios de Windmill primitivos:

$$N_p = \frac{2^{1+2\lfloor L/v \rfloor}}{L} F(L)$$

donde $F(L) = \phi(2^L - 1)/2^L$ si ϕ es la función de Euler.

Podemos concluir sobre este generador que es bastante eficiente para generar secuencias binarias a elevadas velocidades y también para generar secuencias aleatorias en forma de bloque, por lo que sus aplicaciones pueden ser múltiples.

Este generador está propuesto por la *European Broadcasting Union* (EBU) para cifrar imágenes de televisión, usándolo de forma que genere bloques de símbolos pseudoaleatorios.

4.3.2 Generador de permutación de subsecuencias

Las secuencias producidas por este generador [JANSE] se obtienen mediante la permutación de subsecuencias tanto de unos como de ceros de una determinada secuencia binaria periódica $\underline{s} = (s_0, s_1, \ldots, s_{p-1})$ binaria ($s_i \in GF(2)$) de De Bruijn. Las secuencias de De Bruijn de orden n tienen un periodo de 2^n, tal como se vio en el capítulo 3. Esto implica que cada n-tupla ocurre exactamente una vez en la secuencia. De Bruijn [BRU46] demuestra que de dichas secuencias hay exactamente $2^{2^{n-1}-n}$, todas ellas con complejidad de máximo orden [JAN89].

Veamos a continuación cómo obtener familias de secuencias permutando las subsecuencias de unos y ceros de una secuencia de De Bruijn de un orden determinado. El procedimiento consiste en permutar de forma independiente los enteros que representan las subsecuencias de unos y los enteros que representan las subsecuencias de ceros. Una vez hecho esto, se vuelve a transformar la secuencia de enteros obtenida en una secuencia binaria.

El procedimiento descrito preserva las buenas propiedades respecto al cumplimiento del primer y segundo postulado de Golomb de la secuencia original y, además, la secuencia obtenida muestra un comportamiento excelente respecto a la complejidad de máximo orden [JAN89].

Si llamamos C_n a la clase de secuencias obtenidas de esta forma a partir de una secuencia de De Bruijn de orden *n*, el número de secuencias que tendrá esta clase vendrá determinado por la expresión:

$$|C_n| = 2^{-n+2} \prod_{l=1}^{n-2} \left(\frac{2^l}{2^{l-1}} \right)^2$$

Si los coeficientes binomiales de esta expresión se aproximan usando la fórmula de aproximación de Stirling, la ecuación queda como:

$$|C_n| = \rho_n \left(\frac{4}{\pi} \right)^{n-2} \prod_{k=1}^{n} G_k$$

donde $G_k = 2^{2^{k-1}-k}$ es el número de secuencias binarias de De Bruijn de orden *k* y ρ_n es un factor de corrección, menor que 1, independientemente del valor de *n*, y que converge hacia el valor 0,61... para valores de *n* grandes.

De la ecuación anterior se puede deducir que el número de secuencias de De Bruijn contenidas en la clase C_n tenderá a cero a medida que crezca *n*. Sin embargo, todas las secuencias de De Bruijn de orden *n* estarán contenidas en C_n, ya que por definición, están todas aquellas en las que las subsecuencias de longitud *n* ocurren exactamente una vez.

En lo que a la complejidad de máximo orden se refiere, puesto que las secuencias de De Bruijn de orden *n* (y, por tanto, las secuencias con complejidad de máximo orden *n*) son tan sólo una pequeña fracción de C_n, el resto de secuencias de la clase deberán tener necesariamente complejidades de máximo orden mayores que *n*. Se puede afirmar que para todas las secuencias $\underline{s} \in C_n$, $n > 2$, la complejidad de máximo orden $c(\underline{s})$ satisfará la inecuación $n \leq c(\underline{s}) \leq 2^{n-1} - 1$. Además, para $n > 2$ ninguna de ellas puede tener una $c(\underline{s}) = 2^{n-1} - 2$, y para $n \geq 4$ el número de secuencias del conjunto C_n que tendrán complejidad de máximo orden $c(\underline{s}) = 2^{n-1} - 1$ será $2^{-n+5}|C_{n-1}|$.

Veamos cómo generar este tipo de secuencias. Existe una forma eficiente de generarlas mediante el uso de técnicas de codificación enumerativas para generar las permutaciones [COV73]. Sin embargo, a pesar de su sencillez, la implementación de este método suele ser bastante compleja, por lo que a continuación se va a describir otro método más simple basado en LFSRs y contadores binarios que permite generarlas de una forma más sencilla. Este método parte del hecho de que no es necesario generar el conjunto C_n entero, sino tan sólo un subconjunto de éste, con secuencias que puedan aparecer más de una vez (por ejemplo, versiones desplazadas de una secuencia que ya apareció).

La figura 4.23 muestra un ejemplo de generador de permutación de subsecuencias de orden 5, capaz de generar 512 permutaciones diferentes de periodo 32. En dicha figura se pueden identificar dos generadores de De Bruijn de orden 3 basados en registros de desplazamiento realimentados no linealmente, uno para las subsecuencias de unos, y otro para las subsecuencias de ceros. Los

contenidos (estados) de las celdas de estos generadores de De Bruijn se permutan según cuál sea el contenido de los registros de clave.

El vector inicial permite que las dos secuencias de estado producidas se puedan combinar de 8 formas posibles. Las secuencias de estado producidas se pueden entonces convertir en secuencias de enteros mediante la utilización de la caja *S*. Esta caja *S* también se podría implementar de forma separada para ambas secuencias de estados, y permitir entonces la realización de permutaciones adicionales.

Tabla 4.4

Entrada	0	1	2	3	4	5	6	7
Salida	7	7	7	7	6	6	5	3

En la figura anterior, f representa la función no lineal de los generadores de De Bruijn donde $f(x_1,x_2,x_3) = x_1 + x_3 + \overline{x_2 \vee x_3}$, T es una báscula biestable que produce una secuencia de conmutación (10101010...) con una cadencia $c \wedge reloj$, c es la salida de un contador binario de 8 estados, y la caja *S* es la encargada de realizar una permutación de los bits según una tabla de sustitución (tabla 4.4).

Además, los generadores de De Bruijn tienen un reloj de entrada de velocidad $c \wedge reloj \wedge \underline{s}$, donde $\underline{s}$ es la secuencia de salida del generador.

Tabla 4.5

Estado gener. DB	*Entero*	*Gizq*	*Gder*
0 0 0	0	1	2
0 0 1	1	0	3
0 1 1	3	2	0
1 1 1	7	6	7
1 1 0	6	7	1
1 0 1	5	4	5
0 1 0	2	3	4
1 0 0	4	5	6

Veamos con un ejemplo cómo opera este generador. Para ello, se muestra una tabla (tabla 4.5) que contiene los estados sucesivos de los generadores de De Bruijn y los correspondientes valores enteros que servirán como entrada a la caja *S* teniendo en cuenta tanto la clave como el vector inicial. Veamos el proceso por el que se obtiene la secuencia de salida de este generador.

G derecho: 10 . 2 . 6 . . . . 7 . . . . . . 4 . . . 3 . . 5 1
G izquierdo: 2 . 3 . 0 . . . 7 . . . . 1 . . 5 . . 4 . . . 6 . .
c_t: 7 . . . . . 567 34 5 67 3456 7 . 67 6 7 . 6 767 5 6 7 . .
$\{s\}$: 0 1 0 1 0 1 1 1 0 0 0 0 0 1 1 1 1 1 0 1 1 0 0 1 0 0 1 1 0 0

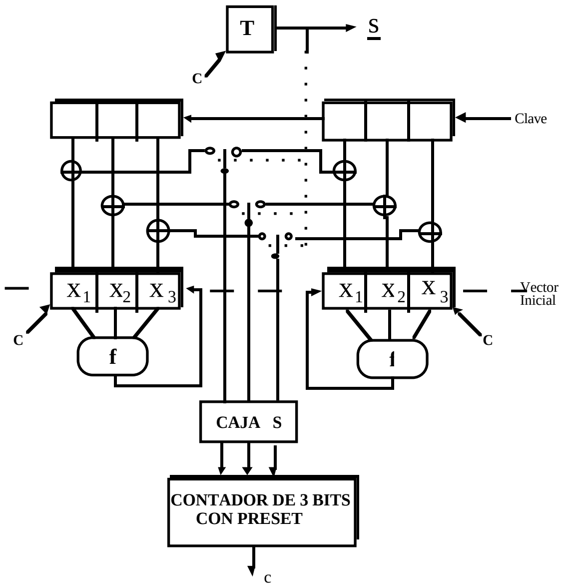


Fig. 4.23 Generador de permutación de subsecuencias

4.3.3 Generador basado en autómatas celulares

Los autómatas celulares han constituído un campo de gran interés desde los primeros estudios realizados por Von Neumann en los años 40. En 1985, Wolfram [WOL86][WOLFR] propuso el uso de autómatas celulares unidimensionales binarios con función de siguiente estado no lineal para generar secuencias pseudoaleatorias. Un autómata celular consiste en *N* celulas dispuestas en línea recta tal como muestra la figura 4.24. Denotemos los valores de las células en el instante *t* como a_1, a_2, ..., a_N, con un valor para cada célula de 1 ó 0. Estos valores se actualizarán en paralelo (sícronamente) en instantes de tiempo bien determinados de acuerdo con una regla fija que tendrá la forma

$$a'(k) = \Phi(a_{k-r}, a_{k-r+1}, \ldots, a_{k+r})$$

donde *r* denota el alcance de los argumentos de entrada a Φ. Cualquier implementación práctica de un autómata celular debe contener un número finito de celulas *N*. Estas presentan las disposiciones típicas en un registro circular con condiciones periódicas en los límites, es decir, los índices de los

argumentos de la regla del siguiente estado Φ se calculan módulo N. Una regla de actualización del siguiente estado interesante es la que viene determinada por $a'(k) = a_{k-1} \oplus (a_k \vee a_{k+1})$.

Veamos cómo funciona el generador de Wolfram. Para ello supongamos que tenemos N células (y posiblemente una función del siguiente estado Φ) cuyo estado inicial es $a(0) = (a_1(0), a_2(0), \ldots, a_N(0))$. El procedimiento es el siguiente:

FOR i = 1, 2,..... DO

Paso 1: Actualizar el autómata celular usando la siguiente regla:

$$a_k(i) = a_{k-1}(i-1) \oplus (a_k(i-1) \vee a_{k+1}(i-1))$$

para valores de k entre 0 y N - 1 y calcular módulo N los índices de las células.

Paso 2: Extraer la secuencia pseudoaleatoria $z(t)$ de cualquier único punto k:

$$z_i = a_k(i) \qquad i = 1, 2, \ldots\ldots$$

END FOR

El diagrama de transiciones entre estados de un autómata celular de N células consiste en un conjunto de ciclos, y se expresa mediante árboles que representan transiciones. Para este generador, el número de estados que tienen dos o más ceros predecesores tiende a cero asintóticamente con N. Para N grandes, el diagrama de transiciones entre estados aparece cada vez más dominado por un único ciclo. La longitud del ciclo máximo Π_N se puede aproximar por $\log_2 \Pi_N \approx 0.61(N + 1)$ para $N < 55$.

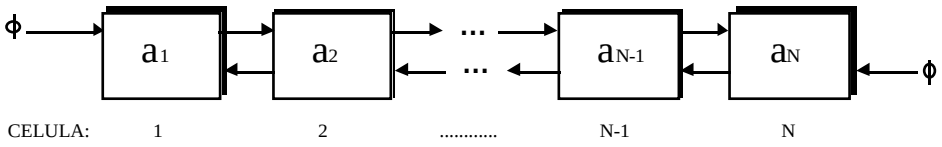


Fig. 4.24 Generador basado en un autómata celular

Además, al someter las secuencias producidas por este generador a una batería de tests estadísticos se observa que no presentan ninguna desviación significativa respecto a las secuencias verdaderamente aleatorias.

4.3.4 Generador de secuencias pseudoaleatorias basado en una memoria

Este generador, propuesto por J.D. Golic [GOLIC] [GOL90], es capaz de producir secuencias pseudoaleatorias mediente una función no lineal variante con el tiempo, aplicada sobre las celdas de

registros de desplazamiento con realimentación lineal (LFSR). Su estructura puede verse en la figura 4.25, donde se observa que está constituído por 3 LFSRs y una memoria de 2^K posiciones de 1 bit. Los tres LFSR deben tener longitudes m_i (i=1,2,3) y polinomios de realimentación primitivos. Los pulsos de reloj se aplican a los tres al mismo tiempo y, si su estado inicial no es el estado todo ceros, generarán m-secuencias de periodos $P_i = 2^{m_i} - 1$ (i=1,2,3). El contenido inicial de la memoria puede ser cualquiera. Las direcciones de *lectura* y *escritura* se tomarán de k celdas del $LFSR_2$ y k celdas del $LFSR_3$ respectivamente, mientras que la salida del $LFSR_1$ se usará para ir llenando la memoria. En cada instante t (t = 0,1,2,...) se producirán las siguientes operaciones:

Paso 1	El bit de salida del generador, $b(t)$, es leído de la posición de la memoria determinada por los k bits obtenidos de las celdas del $LFSR_2$ $x(t)$.
Paso 2	El bit de salida del $LFSR_1$, a(t), se escribirá en la posición de la memoria determinado por los k bits obtenidos de las celdas del $LFSR_3$ $y(t)$.

Debido al estado inicial de la memoria, las secuencias producidas por este generador no tienen por qué ser necesariamente periódicas, ya que presentarán preámbulos. Para hacer que sean periódicas e independientes del contenido inicial de la memoria, asumimos como instante de tiempo inicial $t_0 = P_3$ (de hecho, esto implica no considerar los preámbulos que se producirán antes de que la secuencia entre en una rutina periódica).

Para establecer unas cotas a la periodicidad y a la complejidad de las secuencias producidas por este generador, debemos efectuar las siguientes restricciones: $1 \le k \le \min(m_2, m_3)$ y $2^{m_3} - 1 \le m_1$. La salida del generador $b(t)$ se podrá expresar de la siguiente forma [GOL90]:

$$b(t) = \sum_{s=0}^{P_3-1} C_s(t) V_s(t) \qquad t = 0,1,2,...$$

donde

$$C_s(t) = \begin{cases} 1 & para \quad t - s = 0 \bmod P_3 \\ 0 & para \quad t - s \neq 0 \bmod P_3 \end{cases}$$

con s = 0, 1,..., P_3-1 y $V_s(t) = a(t - \phi_s(x_t))$ con t = 0, 1, 2,... y s = 0, 1,..., P_3-1.

x_t (t = 0,1,2...) es la secuencia de lectura, con periodo P_2, que puede tomar valores en el conjunto $K = \{0,1\}^k$. Para cada s = 0, 1,..., P_3-1, la función $\phi_s(j)$, ($j \in K$), es una aplicación inyectiva $K \rightarrow (1,...,P_3)$ [GOL90].

Si se cumplen las dos restricciones que se habían impuesto anteriormente, y se cumple además que:

a) $mcd(m_1, m_2) \neq m_1$

b) $mcd\left(P_2, \dfrac{P_1}{mcd(P_1, P_2)}\right) = 1$

c) $mcd(P_3, P_1 P_2) = 1$

este generador será capaz de producir $P_1P_2P_3$ secuencias distintas para todos los estados iniciales no nulos de los $LFSR_i$ (i = 1,2,3). El hecho de que este generador sea capaz de producir gran número de secuencias pseudoaleatorias distintas lo hace potencialmente útil para aplicaciones como la criptografía o las comunicaciones *spread spectrum*, que se tratarán más adelante.

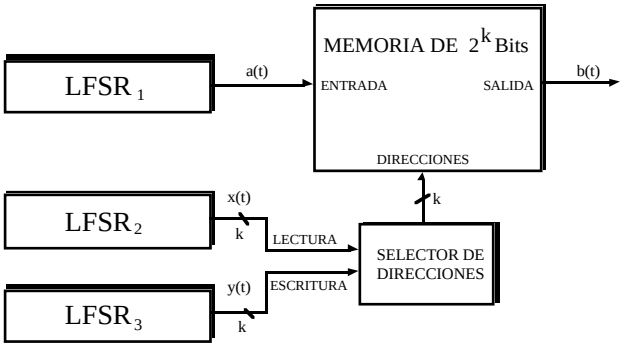


Fig. 4.25 Generador basado en una memoria

4.4. Problemas

PROBLEMA 4.1 Se desea obtener una secuencia pseudoaleatoria de periodo P = 259.967. Establecer los parámetros necesarios si se desea producirla mediante:

a) un generador de Jennings
b) un generador de umbral
c) un generador de Ruepple
d) un generador de marcha y espera de Beth-Piper
e) un generador multivelocidad de Massey-Ruepple
f) un generador de Pless como el de la figura 4.6

(Observar que 259967 = 127 × 2047).

PROBLEMA 4.2 Sea un generador de Geffe, como el de la figura 4.3, con 3 LFSRs de 3, 5 y 7 celdas respectivamente, y todos ellos con polinomio primitivo. Determinar el periodo y la complejidad lineal de la secuencia producida.

PROBLEMA 4.3 Obtener a partir de la secuencia binaria $s(t)$ = 11010111011011000101010011011:

a) una secuencia [2]-autodecimada
b) una secuencia [3]-autodecimada
c) una secuencia [1,2]-autodecimada
d) una secuencia [2,3]-autodecimada

PROBLEMA 4.4 En un generador de Ruepple, como el de la figura 4.9, con 3 LFSRs de 5, 7 y 11 celdas respectivamente y polinomios de realimentación primitivos, determinar:

a) el periodo de la secuencia de salida
b) su grado de inmunidad a la correlación
c) su complejidad lineal

PROBLEMA 4.5 Sea un generador ASG, como el de la figura 4.20, en el que el LFSR-3 se sustituye por un generador de De Bruijn de periodo 256, y los LFSR 1 y 2 tienen 13 y 21 celdas respectivamente y polinomio de realimentación primitivo. Para la secuencia de salida de esta estructura determinar:

a) el periodo
b) la frecuencia de aparición de una subsecuencia de 5 unos
c) la frecuencia de aparición de una subsecuencia de 1 uno
d) la frecuencia de aparición de una subsecuencia de 11 ceros
e) la frecuencia de aparición de una subsecuencia de 1 cero
f) acotar su complejidad lineal

4.5 Bibliografía

[AKLSG] AKL, S. G.; MEIJER, H. *A Fast Pseudo Random Permutation Generator With Applications to Cryptology*. Dept. of comput. and Inform Science. Queens University. Kingston. Ontario, Canada.

[ARA81] ARAZI, B. *On the Synthesis of De Bruijn Sequences*. Information and Control, Vol. 49, Nº 2, May 1981, pp.81-90.

[ARV77] ARVILLIAS, A. C.; MARITSAS, D. G. *Combinational Logicfree Realisations for High-Speed m-sequences Generation*. Electronic Letters, Vol. 13, nº 17, 1977, pp.500-502.

[BET85] BETH, T.; PIPER, F. *The Stop-and-Go Generator*. Advances in Criptology. Proc. Crypt'84. Springer-Verlag Lecture Notes in Computer Science, Nº 209, pp. 88-92, New-York, 1985.

[BLU82] BLUM, L.; BLUM, M.; SHUB, M. *Comparision of Two Pseudo Random Generators*. Proceedings of Crypto 1982.

[BLU84] BLUM, M. MICALLY, S. *How to Generate Cryptographically Strong Seqeunces of Pseudo Random Bits*. SIAM J. Computing 13, 4, pp. 850-864, Nov. 1984.

[BOY89] BOYAR, J. *Interferring Sequences Produced by Pseudo Random Number Generators*. J. ACM, Vol. 36, Nº. 1, Jan. 1989, pp. 129-141.

[BRU46] DE BRUIJN, N. G. A *Combinational Problem*. Nederl-Akad, Wetensch. Proc, Vol 49, pp 758-754, 1946.

[BRU82] BRUER, J.O. *On Nonlinear Combinations of Linear Shift Register Sequences*. Proc. IEEE Int. Symp, Jun. 1982, pp. 21-25.

[CHA84] CHAMB*ERS, W. G.; JENNINGS, S. M. Linear Equivalence of Certain BRM Shift-Register Sequences*. Electron. Lett., Vol. 20, pp. 1018-1019, Nov. 1984.

[CHA88] CHAMBERS, W. G. *Clock-controlled Shift Registers in Binary Sequences Generators.* IEEE Proc. Vol. 135, Pt. E, Nº 1, January 1988.

[CHA88] CHAMBER; GOLLMANN. *Generators for Sequences with Near-Maximal Linear Equivalence*. IEE Proc. Vol 135. Pt. E. Np 1, Jan 1988.

[CHA89] CHAMBERS; GOLLMANN. *Lock-in Effects in Cascades of Clock-Controlled Shift Registers*. Proc. Eurocrypt'88. Springer-Verlag Lecture Notes in Computer Science, Nº 330. New York 1989.

[CHA89] CHAMBERS; GOLLMANN. *Clock-Controlled Shift Registers: A Review*. IEEE J. on Selected Areas in Comm., Vol. 7, Nº 4, May 1989.

[COV73] COVER, T. *Enumerative Source Coding*. IEEE Trans. on Inform. Theory, Vol IT-19, pp. 73-76, January 1973.

[DAV74] DAVIO, M.; BIOUL, G. *Interconnection Structure of Cyclic Counters Made Up of JK Flip-flops*. M.B.L.E. Rep. R279, Brussels, Belgium, Dec. 1974.

[EBU76] *Specification of the Systems of the MAC/packet Family*. European Broadcasting Union. Tech 3258-E (Brussels:EBU Techinical Center), 1976.

[EUR88] EUROCRYPT. *Alternating Steps Generators by De Bruijn Sequences*. Proc. Eurocrypt'87. Springer-Verlag Lecture Notes in Computer Science, Nº 309. New York. 1988.

[FUS91] FUSTER; DE LA GUIA, NEGRILLO, MONTOYA. *Diseño e implementación de algoritmos de Generación de Secuencias Binarias*. I Reunión Española sobre Criptología. Mallorca, Oct. 1991.

[GEF73] GEFFE, P. R. *How to Protect Data with Ciphers that are Really Hard to Break*. Electronics, pp. 99-101, Jan. 4, 1973.

[GOL82] GOLOMB, S. W. *Shift Register Sequences*. Aegean Park Press, Laguna Hills, Calif., 1982.

[GOL84] GOLLMANN, D. *Linear Recursions of Cascaded Sequences.* Contributions to General Algebra, 3, Proc. of the Vienna Conference, June 1984. (Springer-Holder-Pichler-Tempsky, Wien 1985- Verlag B.G. Teuner, Stuttgart).

[GOL85] GOLLMANN, D. *Pseudo-Random Properties of Cascade Connections of Clock-controlled Shift Registers*. Advances in Cryptology. Proc. of Eurocrypt' 84, Vol. 209. Lecture Notes in Computer Science. Beth, Cot, and Ingemarsson Eds. Berlin. Springer-Verlag, 1985.

[GOL90] GOLIC, J. D.; MIHALJEVIC, M. M. *Minimal Binary Sequence Generator.* IEEE Trans. Inform. Theory. vol IT-36, pp. 190-192, Jan. 1990.

[GOL90] GOLIC, J.; MIHALJEVIC, M. M. *Minimal Linear Equivalent Analysis of a Variable-memory Binary Sequence Generator*. IEEE Trns. Inform. Theory, vol. IT-36, pp. 190-192, Jan. 1990

[GOLIC] GOLIC, J. D. *A Number of Output Sequences of a Binary Sequence Generator*.

[GRO71] GROTH. *Generation of Binary Sequences with controllable Complexity*. IEEE Trans. on Information Theory, Vol. 13, Nº 3, May 1971.

[GÜN88] GÜNTHER, C. G. *On the Properties of the Sum of Two Pseudo Random Sequences*. Proc. Eurocrypt'87, Lect. Notes in Comp. Science, Vol. 304, pp. 53-64, Springer-Verlag, 1988.

[GÜN88] GÜNTHER, C. G. *Alternating Step Generators Controlled by De Bruijn Sequences*. Proc. Eurocrypt'88. Springer-Verlag Lecture Notes in Computer Science, Vol. 304, pp. 5-14, New-York, 1988.

[HER82] HERLESTAM, T. *On using Prime Polynomial on Crypto Generators*. Proc. Workshop on Cryptography. Springer-Verlag Lecture Notes in Computer Science, Nº 149, New York, 1982.

[HOP79] HOPCROFT, J. E.; ULLMANN, J. D. *Introduction to Automata Theory*. Languajes and computation Reading, MA. Addison-Wesley, 1979.

[HUR74] HURD, W. J. *Efficient Generation of Statistically Good PseudoNoise by Linearity Interconnected Shift Registers*. IEEE Trans, 1974, C-23, pp.146-152.

[IEE80] IEEE. *Some Randomness Properties of Cascaded Sequences*. Trans. on Inform. Theory, Vol. IT-26, pp. 227-232, Marsch 1980.

[JAN89] JANSEN, C. J. A. *Investigations On Nonlinear Streamcipher systems: Construction and Evaluation Methods.* PhD. Thesis, Technical University of Delft, Delft, April 1989.

[JAN90] JANSEN, C. J. A. *Investigations on Non-Linear Streamciphers Systems: Construction and Evaluation Methods*. Ph. D. Thesis, Thecnical University of Delft, Delft, April 1989.[JANSE] JANSEN, C. J. A. *On the Construction of Run Permutated Sequences*. Philips Crypto B.V. Eindhoven, The Netherlands.

[JEN80] JENNINGS, S. M. *A Special Class of Binary Sequences*. Ph. D. Thesis. Westfield College, London University, 1980.

[JEN82] JENNINGS, S. M. *Multiplexed Sequences: Some Properties of the Minimum Polynolial.* Proc. Workshop on Cryptography, Nº 149. Springer-Verlag Lecture Notes in Computer Science, New York, 1982, pp.189-206.

[KEY76] KEY, E. L. An Analysis of the Structure & Complexity of Non-linear Binary Sequence Generators. IEEE Trans. on Inform. theory, Vol. IT-22, 1976.

[KNO76] KNOTT, G. D. *A Numbering System for Permutations and Combinations*. Communications of the ACM, Vol. 19, Nº 6, June 1976.

[KNU81] KNUTH, D. E. *The Art of Computer Programming. Volume 2: Seminumerical Algorithms.* Addison-Wesley, 1981.

[LEM71] LEMPEL, A.; EASTAMN, W. L. *High Speed Generation of Maximal Lenth Sequences*. IEEE Trans. on Comput., Vol. C-20, 1971, pp. 227-229.

[LID86] LIDL, R..; NEDERREITER, H. *Introduction to Finite Fields and Their Applications*. Cambridge University Press, 1986

[MAC65] MACLAREN, M. D.; MARSAGLIA, G. *Uniform Random Number Generators*. J. Ass. Comput. Machinery, 12, pp. 83-89, 1965.

[MACTE] *Mathematical Foundations of Flip-flops*. MAC Tech. Memo.47.

[MAR77] MARITSAS, D. G.; ARVILLIAS, A. C.; BOUNAS, *A. Phase-Shift Analysis of Linear Feedback Shift-Register Structures Generating Pseudo Random Sequences*. IEEE Transactions, 1977.

[MAS84] MASSEY, J. L.; RUEPPLE, R. A. *Linear Ciphers and Random Sequence Generators with Multiple Clocks*. Proc. Eurocrypt'84. Springer-Verlag Lecture Notes in Computer Science, New-York, 1984.

[MAY87] MAYER, A.; SKJERVEN, T. *Sequences from Self-Clocked Shift-Registers*. Semester Work, ETH-Zürich, 1987.

[MEI92] MEIER; STAFFELBATCH. *Correlation Properties of Combiners with Memory in Stream Ciphers*. J. of Cryptology, Vol. 5, Nº 1, 1992.

[ORE75] O´REILLY, J.J. *Series-parallel Generation of m-sequences*. Radio & Electron. Eng, 1975, 45, pp. 171-176.

[PLE77] PLESS, V. S. Encypting Schemes for Computer confidentiality. IEEE Trans. on Computer, Vol. C-26, Nº 11, 1977.

[PUI92] PUIG, M. *Estudio Comparativo de Cifradores en Flujo. Aplicaciones en Comunicaciones Móviles*. Proyecto final de Carrera. Universidad Politécnica de Cataluña (UPC), 1992.

[QUA74] QUAN, A. Y. C. *A Note on High Speed Generation of Maximal Lenth Sequences*. IEEE Trans., 1974, C-23, pp.201-203.

[RUB79] RUBIN. *Decrypting a Stream Cipher Based on JK flip-flops*. IEEE Trans. on Comput., Vol. C-28, Jul. 1979.

[RUE86] RUEPPLE, R. A. *Analysis and Design of Stream Cipher*. Springer-Verlag, NY, 1986

[RUE88] RUEPPLE, R. A. *When Shift Register Clock Themselves*. Proc. Eurocrypt'87. Springer-Verlag Lecture Notes in Computer Science, Nº 304, New York, 1988.

[SIE84] SIEGENTHALER, T. *Correlation-Inmunity of Non-linear Combinig Functions for Criptographic Applications*. IEEE Trans. on IT, Vol. IT-30, Nº 5, 1984.

[SHA83] SCHAMIR, A. *On the Generation og Cryptographically Strong Pseuod Random Sequences.* ACM Transactions on Computer Systems 1, 1, pp.38-44, Feb. 1983.

[SMEET] SMEETS, B. J. M.; CHAMBERS, W. G. Windmill Generators: *A Generalization and an Observation of How Many There Are*.

[STAFF] STAFFELBACH, O.; MEIER, W. *Analysis of Pseudo Random Sequences Generated by Cellular Automata*.

[SUR78] SURBÖCK, F.; WEINRITCHER, H. I*nterlacing Properties of Shift Register Sequences with Generator Polynomials Irreducible Over GF(p)*. IEEE Trans. on Inform. theory, Vol. IT-24, 1978, pp. 386-389.

[TAT89] TATEBAYASHI, M.; MATSUZAKI, D. B.; NEWMAN, JR. *A Cryptosystem Using Digital Signal Processors for Mobile Communications*. Proc. IEEE, 1989.

[TSA91] TSALIDES, PH.; YORK, T. A..; THANAILAKIS, A. *Pseudorandom Number Generators for CLSI Systems Based on Linear Cellular Automata*. IEE Proceedings-E, Vol. 138, Nº 4, July 1991.

[WOL86] WOLFRAM, S. *Random Sequence Generation by Cellular Automata*. Advances in Applied Mathematics 7, pp.123-169, 1986.

[WOLFR] WOLFRAM, S. *Cryptography with Cellular Automata*. Advances in Cryptology-Crypto'85, Proceedings, pp.429-432, Springer-Verlag, 1986.

[ZEN88] ZENG, K. C.; HUANG, M. Q. *On the Linear Syndrome Method in Cryptanalysis*. Proc. Crypto' 88, Springer-Verlag, Lecture Notes in Computer Science, Nº 403, New-York, 1988.

[ZEN89] ZENG, K. C.; YANG, C. H.; RAO, T. R. N. *On the Linear Consistency Test (LCT) in Cryptanalysis with Applications*. Proc. Crypto' 89, Springer-Verlag, Lecture Notes in Computer Science, Nº 403, New-York, pp. 469-478, 1989.

[ZEN90] ZENG, K. C.; YANG, C. H.; RAO, T. R. N. *Large Primes in Stream-Cipher Cryptography*. Proc. Auscrypt' 90, springer-Verlag Lecture Notes in Computer Science, Nº 453, New-York, pp.194-205, 1990.

[ZEN91] *ZENG; YANG; WEI; RAO. Pseudorandom Bit Generators in Stream-Cipher Criptography*. IEEE Computer. Feb. 1991.

5 Propiedades estadísticas de las secuencias pseudoaleatorias

5.0 Introducción

Si bien es casi siempre difícil desarrollar un análisis matemático de las propiedades de las secuencias pseudoaleatorias, es posible, sin embargo, hacer un estudio empírico y testear su comportamiento con respecto a medidas de aleatoriedad bien conocidas. Esto permite determinar el grado con el que se puede predecir el comportamiento de estas secuencias disponiendo de tan sólo una porción de ellas, o a partir del conocimiento del método de generación.

En este capítulo se expondrán una serie de medidas que se pueden realizar sobre las secuencias pseudoaleatorias que se generen para evaluar su grado de aleatoriedad, es decir, para ver cuánto se parecen a las secuencias generadas mediante el lanzamiento sucesivo de monedas al aire. Para ello, se estudiarán en primer lugar las propiedades de correlación que presentan este tipo de secuencias. Esta función representa una medida importante del grado de dependencia estadística existente entre las diversas partes de la secuencia (función de autocorrelación), o la dependencia existente entre distintas secuencias (función de correlación cruzada).

Si bien en el capítulo 1 ya se estudiaron los conceptos asociados con las secuencias aleatorias, también se verán gran número de propiedades estadísticas asociadas a las secuencias verdaderamente aleatorias, y que servirán para proponer una serie de tests estadísticos que permitirán comprobar hasta qué grado las cumplen las secuencias que se hayan generado. Esto permitirá disponer de una potente herramienta para testear el grado de aleatoriedad de los generadores, ya que todos los tests estadísticos descritos en este capítulo han sido programados y se incluyen con el libro, y puede verse el funcionamiento del programa en el último capítulo.

5.1 Propiedades de correlación de las secuencias pseudoaleatorias

Un forma importante de caracterizar a las secuencias pseudoaleatorias es mediante sus propiedades de correlación [GOL65][MOO68]. Tal como se verá en el capítulo 7, existen muchas aplicaciones en telecomunicaciones que se basan en el uso de estas secuencias, y su eficacia se sustenta en buena parte en las propiedades de correlación que presentan éstas. Algunas de estas aplicaciones no sólo necesitan usar una secuencia, sino una familia de ellas. En este caso, las dependencias que existan entre ellas se convertirá en un punto muy importante que se deberá tener en cuenta, por lo que habrá que establecer unos requisitos que deberán cumplir las secuencias de dicha familia:

1) Cada secuencia del conjunto debe ser fácilmente distinguible de cualquier versión de ella misma desplazada en el tiempo.

2) Cada secuencia del conjunto debe ser fácilmente distinguible de cualquier otra, y de cualquier versión de ellas obtenida mediante un desplazamiento en el tiempo.

El cumplimiento de la primera propiedad es muy importante en aplicaciones como sistemas de radar, comunicaciones de *spread spectrum* y criptografía, mientras que la segunda lo es para aplicaciones de identificación en sistemas con múltiples terminales y sistemas de acceso múltiple por división de código (CDMA). Todas estas aplicaciones se tratarán más detalladamente en el capítulo 7.

Las secuencias que van a usar este tipo de aplicaciones deben ser generalmente periódicas, debido principalmente a la facilidad de implementación del sistema que esto conlleva. Por este motivo, en este apartado sólo se van a ver las propiedades de correlación de este tipo de secuencias, es decir, aquéllas que para algún T cumplen que $x(t) = x(t+T)$ para todo t.

Una forma muy efectiva de medir la distinguibilidad entre dos secuencias es usar una medida basada en el cuadrado de la media, de forma que si la diferencia entre los cuadrados de las medias de dos secuencias es elevada, podremos considerar que son fácilmente distinguibles. Esta medida de distinguibilidad se puede representar como:

$$\frac{1}{T}\int_0^T [y(t) \pm x(t)]^2 dt = \frac{1}{T}\left\{\int_0^T [y^2(t) + x^2(t)]dt \pm 2\int_0^T x(t)y(t)dt\right\}$$

El significado del signo $\pm$ se debe a que no sólo interesa la distinguibilidad entre $y(t)$ y $x(t)$, sino también entre $y(t)$ y $-x(t)$. La primera integral de la segunda parte de la expresión anterior representa la suma de las energías de las señales $x(t)$ e $y(t)$ ($0 \le t \le T$). Por tanto, para energías de las señales fijas, la señal $y(t)$ será fácilmente distinguible de $x(t)$ y de $-x(t)$ si, y sólo si, la magnitud $r = \int_0^T x(t)y(t)dt$ es pequeña. En comunicaciones, sistemas de navegación y radar que emplean receptores basados en la correlación o filtros adaptados, r representa la salida del filtro adaptado a la señal $y(t)$ cuando la entrada es precisamente $x(t)$. En sistemas de acceso múltiple CDMA, $x(t)$ e $y(t)$ pueden representar las señales asignadas a dos transmisores diferentes, tal como se verá en el capítulo 7, y entonces el parámetro r representará la medida de la interferencia (*crosstalk*) entre ambos canales.

Además, según se ha mencianado antes, también se requiere distinguibilidad entre $x(t)$ e $y(t+\tau)$para todo $\tau \in [0,T]$, si $x(t)$ e $y(t)$ son señales diferentes, y para todo $\tau \in (0,T)$ si $x(t)$ e $y(t)$ son la misma señal. Esto nos lleva proponer como medida de interés la que proporciona la función de correlación cruzada [SAR80][HEL76]:

$$\tau_{x,y}(t) = \int_0^T x(t)y(t+\tau)dt$$

Hasta aquí se ha considerado que tanto $x(t)$ como $y(t)$ son señales reales. Si se considerasen señales complejas, bastaría modificar la ecuación anterior sustituyendo $y(t)$ por su conjugada compleja.

Sin embargo, debido a la naturaleza digital de los sistemas que generalmente emplean las secuencias pseudoaleatorias, el análisis de la distinguibilidad que nos interesa es el que hace referencia a las señales de tipo discreto. Las señales que se usarán en la práctica consistirán en secuencias de pulsos elementales limitados en el tiempo de forma que:

$$x(t) = \sum_{n=-\infty}^{\infty} x_n \delta(t - nT_c)$$

donde $\delta(t)$ es una foma de onda con forma de pulso y T_c es la duración de dicho pulso. Si $x(t) = x(t+T)$ para todo t, entonces T debe ser un múltiplo de T_c, y la secuencia discreta $\{x_n\}$ deberá ser periódica con un periodo que será divisor de $N = T/T_c$. Por tanto, trataremos ahora secuencias discretas periódicas que representaremos por x_n e y_n.

Podemos definir la función de correlación periódica entre las secuencias $\{x_n\}$ e $\{y_n\}$ como:

$$C_{x,y}(l) = \sum_{n=0}^{N-1} x_n y_{n+l}$$

Se puede ver que $r_{x,y}(\tau) = \lambda C_{x,y}(l)$ siempre que $\tau = lT_c$ y donde $\lambda = \int_0^{T_c} \delta^2(t)dt$.

Debido a la importancia que tienen las *m*-secuencias en la mayoría de aplicaciones que hacen uso de este tipo de secuencias, en el apartado siguiente se muestra su comportamiento respecto a las diversas medidas de correlación.

5.1.1 Propiedades de correlación de las *m*-secuencias

El problema de obtener conjuntos de secuencias pseudoaleatorias periódicas con buenas propiedades de correlación se reduce a diseñar conjuntos de secuencias que cumplan con las siguientes propiedades:

1) Para cada secuencia $x = \{x_n\}$ del conjunto, el módulo de su función de autocorrelación $|C_{x,x}(l)|$ debe ser pequeño para $1 \le l \le N-1$.
2) Para cada par de secuencias $x = \{x_n\}$ e $y = \{y_n\}$ del conjunto, el módulo de su correlación cruzada $|C_{x,y}(l)|$ deberá ser pequeño para todo l.

Tal como se vio en el capítulo 3, uno de los tipos de secuencias pseudoaleatorias más importantes son las *m*-secuencias, obtenidas a partir de registros de desplazamiento con realimentación lineal (LFSR). Se vio también que si n era el número de celdas del LFSR y el polinomio de realimentación era primitivo, su periodo era $N = 2^n-1$. Una de las principales características de las *m*-secuencias es que presentan una función de autocorrelación bivalor de forma que $C_{x,x}(0) = N$ y $C_{x,x}(l) = -1$ para $1 \le l < N$. Esta es la mejor autocorrelación posible que se podrá obtener con secuencias binarias de periodo N, es decir, los menores posibles valores de $|C_{x,x}(l)|$ para $1 \le l < N$ (lóbulos de autocorrelación mínimos). Esta propiedad de autocorrelación ideal que presentan las *m*-

secuencias es apropiada en un buen número de aplicaciones, tal como se verá en el capítulo 7. Recordemos, además, que el tercer postulado de Golomb, que se vio en el capítulo 1, comprobaba precisamente el cumplimiento de esta propiedad [GOL67]. Veamos algunas cotas de la función de autocorrelación de secuencias pseudoaleatorias periódicas. Para un conjunto de ξ secuencias periódicas, podemos definir la magnitud de pico de la correlación cruzada como:

$$C_c = max\{|C_{x,y}(l)|: 0 \le l \le N, x \in \xi, y \in \xi, x \ne y\}$$

y la magnitud de pico de la autocorrelación fuera de fase como:

$$C_a = max\{|C_{x,x}(l)|: 1 \le l \le N-1, x \in \xi\}$$

Sidelnikov y Welch [SID69] [WEL74] demostraron que si el conjunto ξ contiene K secuencias, entonces:

$$C_{max} \equiv max\{C_a, C_c\} \ge N\left[\frac{K-1}{NK-1}\right]^{\frac{1}{2}}$$

Veamos del mismo modo algunas propiedades de la función de correlación cruzada entre dos *m*-secuencias. Para ello, consideremos las *m*-secuencias $\{x_n\}$ e $\{y_n\}$, ambas con periodo $N = 2^n-1$. La correlación cruzada entre estas dos *m*-secuencias también será periódica de periodo N, $C_{x,y}(l) = C_{x,y}(l+N)$, y se cumplirá que $|C_{x,y}(l)| \le N$ para todo l. La función $C_{x,y}(l)$ será siempre un entero impar. Helleseth [HEL76] demuestra que para todo l, $C_{x,y}(l) + 1$ es múltiplo de 8, es decir, $C_{x,y}(l) = -1 \mod 8$, excepto en el caso de que x e y estén generados usando polinomios recíprocos, en cuyo caso $C_{x,y}(l) + 1$ será múltiplo de 4. Además demuestra que:

$$\sum_{l=0}^{N-1} C_{x,y}(l) = +1$$

de donde se puede deducir que para valores del periodo N grandes, el valor medio de $C_{x,y}(l)$ será muy cercano a cero. Otra propiedad sobre la función de correlación cruzada de *m*-secuencias es que cumple:

$$\sum_{l=0}^{N-1} [C_{x,y}(l)]^2 = N^2 + N - 1 = 2^{2n} - 2^n - 1$$

de donde se deduce que el valor cuadrado medio de $C_{x,y}(l)$ estará muy cercano a 2^n, y que $|C_{x,y}(l)|$ excederá $2^{n/2} - 1$ para al menos un valor de l. Sin embargo, Sidelnikov [SID69] da una cota más restrictiva que implica que para al menos un entero l, se cumple que:

$$|C_{x,y}(l)| > -1 + 2^{(n+1)/2} \qquad (1)$$

En la mayoría de aplicaciones, no será necesario conocer el valor de $C_{x,y}(l)$ para cada l. Frecuentemente no se necesitará saber más que el conjunto de valores de correlación cruzada y los enteros l ($0 \le l < N$) para los cuales $C_{x,y}(l) = c$, para cada c de este conjunto. Este valor, así definido, se denomina espectro de la función de correlación cruzada del par de secuencias (x,y), y generalmente suele ser mucho más fácil de calcular que la función de autocorrelación. Pero ¿qué forma tiene este espectro?. El espectro de la función de autocorrelación será una función bivalor, donde el valor N ocurrirá una vez y el valor -1 ocurrirá N - 1 veces. Para la función de correlación cruzada, Golomb [GOL68] demostró que si las secuencias $\{x\}$ e $\{y\}$ son dos m-secuencias generadas por polinomios primitivos distintos, su espectro tendrá al menos tres valores. Si se define $t(n)$ como $t(n) = 1 + 2^{\lfloor (n+2)/2 \rfloor}$ (donde $\lfloor p \rfloor$ representa la parte entera del número real p), y si $n \neq 0 \bmod 4$, entonces existirán parejas de m-secuencias con funciones de correlación cruzada con tres valores que serán -1, -$t(n)$ y $t(n)$-2 [PUR77]. A este tipo de función se la denomina función de correlación cruzada preferente con tres valores, y a las m-secuencias que las producen pares preferentes de m-secuencias.

Si C_c denota la máxima magnitud de la correlación cruzada para un determinando par de m-secuencias $\{x\}$ e $\{y\}$, ambas de periodo 2^n-1, $n \ge 3$, se puede concluir que:

1) Cuando n es impar, o cuando $n = 2 \bmod 4$, $C_c = t(n)$ para parejas preferentes de m-secuencias.
2) Cuando n es par, $C_c = t(n)$-2 para m-secuencias generadas con polinomios recíprocos.
3) Cuando n es un múltiplo de 4, $C_c = t(n)$-2 para parejas de m-secuencias de periodo 2^n-1 donde n sea un múltiplo de 4.

Cuando n sea impar, la cota definida en (1) junto con el hecho de que C_c sea un entero impar, implicará que $C_c \ge t(n)$ para cualquier par de m-secuencias. Cuando n sea par, $C_c \ge t(n)$-2 para cualquier par de m-secuencias.

Si bien hasta aquí se han analizado las propiedades de correlación que presentan las m-secuencias, a la hora de diseñar generadores de secuencias pseudoaleatorias debe ser un objetivo fundamental el que las secuencias que éstos generen, tengan un comportamiento lo más aproximado posible al que presentan éstas. Esto, tal como se verá en el capítulo 7, es fundamental para ciertas aplicaciones como el cifrado en flujo. Sin embargo, para otras aplicaciones de telecomunicaciones, como la modulación en *spread spectrum* o el acceso múltiple por división de código (CDMA), el diseño de los conjuntos de secuencias periódicas requiere que éstas cumplan además otros requisitos que afectan a las propiedades de la correlación no periódica. Esta propiedad se verá en el apartado siguiente.

5.1.2 La función de correlación aperiódica (impar)

En muchas aplicaciones que emplean secuencias periódicas, cada una de ellas está modulada de alguna forma por una señal de datos. Una forma muy común de hacer esto, es que una señal de datos binarios determine la polaridad de la secuencia binaria. Esto puede verse como una modulación binaria de amplitud (ASK) que se usa en sistemas de s*pread-spectrum* basados en secuencia directa. Sin embargo, en muchos sistemas hay más de una de estas señales moduladas transmitidas al mismo tiempo y en la misma banda de frecuencia. En general, la única forma de hacer distinguibles para el receptor estas señales es asignar a cada transmisor una única secuencia llamada secuencia de firma.

Estas secuencias, tal como se verá en el capítulo 7, deben tener buenas propiedades de correlación. Para estudiar las propiedades de este tipo de secuencias, suele ser de gran utilidad definir la correlación no periódica (o también aperiódica ó impar) entre dos secuencias complejas $\{x\}$ e $\{y\}$, ambas de periodo N, como:

$$CA_{x,y} = \begin{cases} \sum_{j=0}^{N-1-l} x_j y_{j+1}^* & 0 \le l \le N-1 \\ \sum_{j=0}^{N-1+l} x_{j-l} y_j^* & 1-N \le l < 0 \\ 0 & |l| \ge N \end{cases}$$

En el origen, el valor de la correlación periódica vale lo mismo que la aperiódica, es decir, $CA_{x,y}(0) = C_{x,y}(0)$.

Al igual que ocurría con la correlación periódica, también podemos definir unas cotas para la función de correlación aperiódica. Si tenemos un conjunto de ξ secuencias periódicas, podemos definir la magnitud de pico de la correlación cruzada aperiódica como:

$$CA_c = max\left\{\left|CA_{x,y}(l)\right| : 0 \le l \le N, x \in \xi, y \in \xi, x \ne y\right\}$$

y la magnitud de pico de la autocorrelación aperiódica como:

$$CA_a = max\left\{\left|CA_{x,x}(l)\right| : 1 \le l \le N-1, x \in \xi\right\}$$

Si el conjunto contiene K secuencias, se puede acotar inferiormente el valor de la correlación aperiódica mediante la expresión:

$$CA_{max} \equiv max\left\{CA_a{}^2, CA_c{}^2\right\} \ge \frac{N(K-1)}{(2NK-K-1)}$$

La correlación aperiódica juega un papel fundamental en el diseño y análisis de secuencias periódicas para ciertos sistemas de acceso múltiple para comunicaciones, como CDMA y para sistemas de modulación de *spread spectrum*, que se describirán en el capítulo 7.

5.2 Propiedades estadísticas de las secuencias pseudoaleatorias

Si bien la aleatoriedad es una propiedad asociada conceptualmente a las secuencias de longitud infinita, las secuencias que generalmente se obtienen mediante el uso de dispositivos electrónicos son periódicas y, por tanto, de longitud finita, por lo que es necesario hacer extensible algunas propiedades de aleatoriedad de las secuencias infinitas a este tipo de secuencias para poder estudiarlas. Para ello, se utilizan una serie de tests que nos permiten detectar los defectos estadísticos que pueda presentar la

secuencia pseudoaleatoria bajo estudio. En la práctica, se suele usar una batería de 10 ó 12 de estos tests estadísticos. Sin embargo, cuantos más apliquemos sobre la secuencia, más información obtendremos sobre su comportamiento aleatorio, o hasta qué punto cumple la propiedad de aleatoriedad, lo que nos permitirá rechazar al generador que la produjo si falla al pasar la mayoría de ellos. Se describen por tanto los tests estadísticos más útiles, y se muestran los algoritmos que los implementan y qué tipo de defecto revela cada uno de ellos.

5.2.1 Fundamentos de los tests estadísticos

Los tests estadísticos se usan para detectar posibles defectos estadísticos en los generadores de bits aleatorios. Para ello, examinan una muestra de la secuencia de una cierta longitud *N,* y rechazan el generador que la produjo cuando ciertas propiedades de la muestra de la secuencia indican una posible no aleatoriedad (por ejemplo, que el número absoluto de ceros y unos difiera considerablemente). Veamos el fundamento de este tipo de tests.

5.2.2 Tests de hipótesis

A la hora de estudiar el comportamiento estadístico de las secuencias producidas por un generador, generalmente no tendremos ninguna información preliminar sobre cuál va a ser la distribución que va a seguir. La única forma de abordar este problema es considerar el bit de salida del generador como una variable aleatoria y realizar diversas observaciones sobre ella para obtener una muestra que nos permita caracterizar el comportamiento de dicha variable aleatoria.

En el caso de que no tuviéramos ninguna referencia de que la variable aleatoria que estudiamos siga alguna de las distribuciones conocidas, lo mejor será construir una distribución empírica. Sin embargo, a la hora de estudiar las secuencias producidas por generadores pseudoaleatorios, es de esperar que estas secuencias cumplan una serie de distribuciones que sí se cumplirían en el caso de secuencias verdaderamente aleatorias. En este supuesto, es mejor hacer la hipótesis de que la variable aleatoria seguirá una determinada distribución (hipótesis estadística), y usar métodos estadísticos para validar dicha hipótesis. Una vez que se ha averiguado la distribución que sigue la variable aleatoria, será necesario, además, determinar los parámetros de esta distribución. Por ejemplo, si es una distribución normal, estos parámetros serán su media y su varianza.

Básicamente, un test de hipótesis consiste en una regla sencilla que permite rechazar o aceptar la hipótesis que hemos realizado sobre la distribución de la variable aleatoria. Esta decisión, se basa en ciertas estadísticas que obtendremos al examinar una cierta muestra o conjunto de muestras de la secuencia. Además, un parámetro importante del test estadístico serán los valores de la regla para los cuales se producirá un rechazo de la hipótesis.

Obviamente, mediante el empleo de esta metodología, podría darse el caso de que la muestra estudiada nos llevara a rechazar una hipótesis correcta. A este tipo de error se le suele denominar de tipo I y la probabilidad de que ocurra se le llama nivel de significancia α. Otra posibilidad, es que la muestra estudiada nos llevara a aceptar una hipótesis que fuese incorrecta, lo que produciría un error de tipo II.

La probabilidad de que se produzca un error de este tipo se suele denotar por β. Un resumen de estos posibles casos a la hora de realizar el test de hipótesis se puede ver en la tabla 5.1, para una determinada hipótesis *H* sobre la distribución de la variable aleatoria que se analiza.

Tabla 5.1 Posibles resultados del test de hipótesis

Resultado de Test	*Situación real*	
	H es falsa	*H es correcta*
H rechazada	Decisión correcta	Error de tipo I
H aceptada	Error de tipo II	Decisión correcta

Uno de los objetivos que se deberá tener en cuenta a la hora de realizar el test de hipótesis será minimizar α y β, es decir, las probabilidades de que nos equivoquemos a la hora de decidir sobre la validez o no de la hipótesis que hemos realizado. Desafortunadamente, si disminuímos una de las dos probabilidades, la otra aumentará, y la única manera de hacer que el riesgo de cometer ambos errores disminuya, consiste en basar nuestra decisión en una muestra lo más grande posible. La mayoría de las veces, lo mejor será fijar a un valor aceptable α y fijar la regla de decisión de forma que minimice β. Podemos definir una metodología general para realizar un test de hipótesis de la siguiente forma:

Paso 1: Establecer una hipótesis que se luego se comprobará.

Paso 2: Establecer un riesgo aceptable de rechazar esta hipótesis suponiendo que fuera correcta (es decir, asignar un valor a α).

Paso 3: Elegir un test estadístico adecuado para comprobar la validez de la hipótesis *H*.

Paso 4: Asumir que la hipótesis *H* es correcta y determinar la distribución que cumplirá el test estadístico.

Paso 5: Determinar una región crítica en la cual la hipótesis *H* se rechace el 100% de las veces cuando α sea verdadera.

Paso 6: Capturar mediante la observación una muestra de la secuencia que se estudia de un determinado tamaño, y calcular a partir de ella el test estadístico y el test de hipótesis.

Si bien existen diversos métodos para realizar el test de hipótesis, como el test-*t* [GRAYB], el test-*F* [GRAYB] o el test de bondad de aproximación chi-cuadrado [KNU67], nos vamos a centrar en este último, ya que la mayoría de tests estadísticos se basan en él.

A) El test de bondad de aproximación chi-cuadrado (χ^2)

Este test [KNU67] consiste en un método básico de muestreo estadístico que se usa en conexión con otros tests. Sin embargo, su aplicación supera ampliamente la mera inspección de secuencias, ya que su utilidad se extiende a otras muchas áreas que necesitan de la estadística, como la sociología, el marketing etc. El test chi-cuadrado realiza una comparación entre los valores observados y los valores esperados para la distribución de un conjunto de observaciones (muestra) realizadas sobre la variable aleatoria que estudiamos. La comprobación de la hipótesis de que la distribución esperada (E) y la observada (Y) a partir de una muestra consistente en *n* observaciones serán la misma se realiza mediante el test estadístico:

$$\chi^2 = \sum_{i=1}^{n} \frac{(Y_i - E_i)^2}{E_i}$$

Veamos cómo funciona mediante un ejemplo. Se tienen dos dados y se quiere comprobar si están o no trucados. Para ello se realiza un experimento consistente en lanzar sucesivamente los dos dados suponiendo a priori que éstos no están trucados, es decir, que son independientes y con lados equiprobables. Lo que pretendemos es averiguar qué distribución seguirá la suma *s* de los valores observados con ambos dados al lanzarlos un número determinado de veces *n*, y compararla con la distribución que seguirían si realmente no estuvieran trucados, y que podemos calcular teóricamente con facilidad. La siguiente tabla muestra estas probabilidades esperadas para esta suma de los dos dados:

valor de s =	2	3	4	5	6	7	8	9	10	11	12
probabilidad p_S =	1/36	1/18	1/12	1/9	5/36	1/6	5/36	1/9	1/12	1/18	1/36

Si se lanzan los dados *n* veces, debería obtenerse un valor determinado de la suma np_S veces en media. Supongamos que se tiran los dos dados 144 veces ($n = 144$) y que se obtienen los siguientes resultados:

	valor de s =	2	3	4	5	6	7	8	9	10	11	12
valores	observados Y_S =	2	4	10	12	22	29	21	15	14	9	6
valores	esperados nP_S =	4	8	12	16	20	24	20	16	12	8	4

(2)

Si se realizan 144 lanzamientos de los dos dados, se podrán obtener 36^{144} secuencias posibles y todas igualmente probables. ¿Cómo se podría saber si los dados están o no trucados? Una de las secuencias anteriores será la secuencia formada por 144 doses.

Sin embargo, si saliese esta secuencia, probablemente se pensaría que los dados están trucados, aunque sea tan probable como cualquier otra. No se puede, pues, responder a esto con un sí o no definitivo, pero sí se puede dar una respuesta probabilística que se base en decir cuán probables o improbables son ciertos eventos. Para ello, se puede calcular la estadística chi-cuadrado de las cantidades observadas Y_2,..., Y_{12} en el experimento considerado, que para los valores obtenidos en (2) valdrá:

$$\chi^2 = \frac{(2-4)^2}{4} + \frac{(4-8)^2}{8} + \ldots + \frac{(6-4)^2}{4} = 7,14583$$

Una manera de proceder sería considerar los cuadrados de las diferencias entre los valores observados y los valores esperados, es decir:

$$\chi^2 = (Y_2 - nP_2)^2 + (Y_3 - nP_3)^2 + \ldots + (Y_{12} - nP_{12})^2 \qquad (3)$$

Si el par de dados no es bueno, el valor de χ^2 debería salir relativamente elevado. Pero, para cualquier valor de χ^2 nos podemos preguntar: ¿Cuál es la probabilidad de que χ^2 sea elevado? Si esta probabilidad es muy pequeña, por ejemplo 1/100, sabremos que sólo una vez entre 100 los dados darán unos valores alejados de los esperados, por lo que habrán motivos fundados de sospecha. Notar que incluso unos dados buenos podrían dar un valor alto de χ^2 una de cada 100 veces, por lo que una persona cauta repetiría el experimento para ver si este valor elevado de χ^2 se repite.

La estadística de (3) da un peso igual a todos los elementos pero, por ejemplo, el 7 ocurre aproximadamente seis veces más frecuentemente que el 2.

Por tanto, una buena estadística sería aquella que diese al componente $(Y_7 - nP_7)^2$ del sumatorio de la expresión (3) un sexto de la importancia que se diera a $(Y_2 - nP_2)^2$, por lo que se tendría:

$$\chi^2 = \frac{(Y_2 - nP_2)^2}{nP_2} + \frac{(Y_3 - nP_3)^2}{nP_3} + \ldots + \frac{(Y_{12} - nP_{12})^2}{nP_{12}} \qquad (4)$$

La cuestión ahora es determinar si el valor 7,14583 obtenido es un valor elevado de χ^2 improbable. Para ello, se recurrirá a la utilización del método general chi-cuadrado.

Paso 1	Realizar *n* observaciones independientes de la variable aleatoria que se quiere estudiar y realizar una tabla de frecuencia a partir de los valores obtenidos (i, Y_i, para $1 \leq i \leq n$) de las observaciones. Cada observación podrá caer dentro de una de *k* categorías posibles.
Paso 2	Calcular las frecuencias de aparición teóricas o esperadas (E_s) para cada intervalo *s*, asumiendo que la hipótesis de distribución considerada es correcta. Esta distribución esperada se puede obtener a partir de las probabilidades P_s de que cada observación caiga dentro de la categorías de forma que $E_S = nP_s$.
Paso 3	Si Y_s son el número de observaciones que han caído dentro de la categoría *s*, calcular las cantidades $(Y_s - nP_s)^2 / nP_s$, para $s = 1...k$.
Paso 4	Cacular la estadística chi-cuadrado mediante la expresión:

$$\chi^2 = \sum_{s=1}^{k} \frac{(Y_s - nP_s)^2}{nP_s}$$

Los grados de libertad para esta estadística serán $v = k$-1.

Paso 5 Elegir un valor para α y realizar el test de hipótesis, rechazando que la hipótesis que se había asumido y la obtenida a partir de las muestras son la misma si:

$$\chi^2 \geq \chi^2(1-\alpha, v)$$

donde $\chi^2(1-\alpha, v)$ es el valor que obtiene la distribución χ^2 para v grados de libertad y un nivel de significancia α.

Puesto que $(Y_S - nP_S)^2 = Y_S^2 - 2nP_SY_S + n^2P_S^2$ y usando el hecho de que:

$$\begin{aligned} Y_1 + Y_2 + ... + Y_k &= n \\ P_1 + P_2 + ... + P_k &= 1 \end{aligned} \qquad (5)$$

la expresión del test chi-cuadrado de (4) se puede calcular más fácilmente mediante la expresión:

$$\chi^2 = \frac{1}{n} \sum_{s=1}^{k} \left(\frac{Y_s^2}{P_s} \right) - n$$

Para realizar el paso 5 será preciso ayudarse de una tabla (tabla 5.2), que muestra los valores de la distribución chi-cuadrado[1] para diferentes grados de libertad v. La fila que se debe usar es la de $v = k$-1, ya que el número de grados de libertad es uno menos que el número de categorías (intuitivamente esto es porque $Y_1, Y_2,..., Y_k$ no son totalmente independientes, ya que según se ve en la ecuación (5) se puede calcular una de ellas si se conocen las demás.

El mecanismo de la tabla se puede establecer de la siguiente manera: si se entra en la fila con v grados de libertad y la columna con probabilidad p, y en esa posición está un número x, χ^2 será mayor que x con probabilidad p.

Otra cuestión importante se refiere a cuál debe ser el tamaño de n. Una buena regla es tomar n lo suficientemente grande como para que cada uno de los valores esperados nP_S valga 5 o más. Observar que en el ejemplo visto nP_2 vale sólo 4, violando esta regla.

Una última cuestión interesante sobre este test es si conviene o no repetirlo diversas veces, es decir, ¿tendremos suficientes garantías con los resultados obtenidos de aplicar un solo test?. Lo cierto es que estaremos más seguros si somos capaces de repetir el test, ya que no olvidemos que el resultado que nos ofrece es probabilístico (la probabilidad de que suceda como mínimo χ^2 es p). Además, si

repetimos el test estaremos investigando en otro tramo de la secuencia escogida. ¿Qué ocurrirá si en un determinado conjunto de tests chi-cuadrado, una parte da resultados positivos y otra da resultados negativos? ¿Cómo decidiremos si merece nuestra confianza?. Knuth [KNU67] propuso pasar esos resultados por otro test: el test de Kolmogorov-Smirnov que se describe a continuación.

Tabla 5.2 Valores seleccionados de la distribución chi-cuadrado

	$p = 99\%$	$p = 95\%$	$p = 75\%$	$p = 50\%$	$p = 25\%$	$p = 5\%$	$p = 1\%$
$\nu = 1$	0,00016	0,00393	0,1015	0,4549	1,323	3,841	6,635
$\nu = 2$	0,00201	0,1026	0,5753	1,386	2,773	5,991	9,210
$\nu = 3$	0,1148	0,3518	1,213	2,366	4,108	7,815	11,34
$\nu = 4$	0,2971	0,7107	1,923	3,357	5,385	9,488	13,28
$\nu = 5$	0,5543	1,1455	2,675	4,351	6,626	11,07	15,09
$\nu = 6$	0,8720	1,635	3,455	5,348	7,841	12,59	16,81
$\nu = 7$	1,239	2,167	4,255	6,346	9,037	14,07	18,48
$\nu = 8$	1,646	2,733	5,071	7,344	10,.22	15,51	20,09
$\nu = 9$	2,088	3,325	5,899	8,343	11,39	16,92	21,67
$\nu = 10$	2,558	3,940	6,737	9,342	12,55	18,31	23,21
$\nu = 11$	3,053	4,575	7,584	10,34	13,70	19,68	24,73
$\nu = 12$	3,571	5,226	8,438	11,34	14,84	21,03	26,22
$\nu = 15$	5,229	7,261	11,04	14,34	18,25	25,00	30,58
$\nu = 20$	8,260	10,85	15,45	19,34	23,83	31,41	37,57
$\nu = 30$	14,95	18,49	24,48	29,34[1]	34,80	43,77	50,89
$\nu = 50$	29,71	34,76	42,94	49,33	56,33	67,50	76,15
$\nu > 30$	*aproximadamente* $\nu + 2\sqrt{\nu}\,x_p + \frac{4}{3}x_p^2 - \frac{2}{3}$						
x_p	-2,33	-1,64	-0,675	0,00	0,675	1,64	2,33

B) El test de Kolmogorov-Smirnov (KS)

Este test sirve para dar una idea de cómo se parecen dos curvas, una continua y conocida en todos los puntos $F(x)$, y otra obtenida por muestras de algún proceso estadístico que presuntamente debería seguir la misma curva $F_N(x)$. Este test nos da la probabilidad de que ocurra la máxima diferencia encontrada entre ambas. A diferencia de lo que ocurre con el test chi-cuadrado, este test no

[1] Más valores de la distribución chi-cuadrado se pueden encontrar en la tabla 26.8 del *Handbook of Mathematical Functions*, editado por M. Abramowitz e I.A. Stegun para la Oficinal Editorial de Gobierno de los Estados Unidos.

está sujeto a restricciones según cuál sea el valor de *n*, y para cada *n* se puede encontrar esa probabilidad con la misma fiabilidad.

Si se realizan *N* observaciones independientes de una variable aleatoria X, y se obtienen los valores $X_1, X_2,..., X_N$, se puede crear la función de distribución empírica $F_N(x)$:

$$F_N(x) = (\text{número de } X_1, X_2,..., X_N \text{ que son} \le x)/N$$

A medida que *N* crezca, $F_N(x)$ será una mejor aproximación a la función de distribución real $F(x)$. Los generadores de secuencias malos darán funciones de distribución empíricas que no se aproximarán a $F(x)$ demasiado bien. Para realizar este test, se deben calcular las siguientes estadísticas:

$$K_N^+ = \sqrt{N}\, max_{-\infty<x<\infty}(F_N(x) - F(x)) \qquad K_N^- = \sqrt{N}\, max_{-\infty<x<\infty}(F(x) - F_N(x)) \qquad (6)$$

donde K_N^+ mide la desviación máxima cuando F_N es mayor que F y K_N^- mide la desviación máxima cuando F_N es menor que F. El factor $\sqrt{N}$ se pone porque para *x* fijos de forma que la desviación estándar de $F_N(x)$ respecto a $F(x)$ es proporcional a $1/\sqrt{N}$. Por tanto, este factor magnifica las estadísticas K_N^+ y K_N^- de forma que sean independientes de *N*. Al igual que ocurría en el test chi-cuadrado, podemos comparar K_N^+ y K_N^- en una tabla de porcentajes (tabla 5.3) para ver si son significativamente altos o bajos. Como las estadísticas definidas en (6) no están adaptadas para los cálculos mediante un ordenador, ya que buscan un máximo entre una gran cantidad de valores (*x*!) y el hecho de que $F(x)$ es creciente y de que $F_N(x)$ se incrementa solamente en pasos finitos, hace conveniente ver una forma más simple para evaluar las estadísticas de K_N^+ y K_N^-. Los pasos son:

Paso 1 Realizar *N* observaciones observaciones $X_1, X_2,..., X_N$ y almacenarlas en un vector.

Paso 2 Reordenar el vector en orden ascendente: $X_1 \le X_2 \le ... \le X_N$.

Paso 3 Calcular j/N ($0 \le j \le N$) y guardarlo en otro vector. Estos son los valores con los que se compararán las funciones.

Paso 4 Obtener las estadísticas mediante las fórmulas:

$$K_N^+ = \sqrt{N}\, max_{1\le j\le N}\left(\frac{j}{N} - F\left(x_j\right)\right) \qquad K_N^- = \sqrt{N}\, max_{1\le j\le N}\left(F\left(x_j\right) - \frac{j-1}{N}\right) \qquad (7)$$

Paso 5 Calcular la probabilidad con que se obtendrá K_N^+ mediante:

$$P\left(K_N \le \sqrt{t(n)}\right) = {}^{t}\!/_{N^N} \sum_{0\le k\le t} \binom{N}{k} (k-t)^k (t+N-k)^{N-k-1}$$

Tabla 5.3 Valores seleccionados de las distribuciones K_N^+ *y* K_N^-

	p = 99%	*p = 95%*	*p = 75%*	*p = 50%*	*p = 25%*	*p = 5%*	*p = 1%*
n = 1	0,01000	0,05000	0,2500	0,5000	0,7500	0,9500	0,9900
n = 2	0,10400	0,06749	0,2929	0,5176	0,7071	1,0980	1,2728
n = 3	0,01699	0,07919	0,3112	0,5147	0,7539	1,1017	1,3589
n = 4	0,01943	0,08789	0,3102	0,5110	0,7642	1,1304	1,3777
n = 5	0,02152	0,09471	0,3249	0,5245	0,7674	1,1392	1,4024
n = 6	0,02336	0,1002	0,3272	0,5319	0,7703	1,1463	1,4144
n = 7	0,02501	0,1048	0,3280	0,5364	0,7755	1,1537	1,4246
n = 8	0,02650	0,1086	0,3280	0,5392	0,7797	1,1586	1,4327
n = 9	0,02786	0,1119	0,3274	0,5411	0,7825	1,1624	1,4388
n = 10	0,02912	0,1147	0,3297	0,5426	0,7845	1,1658	1,4440
n = 11	0,03028	0,1172	0,3330	0,5439	0,7863	1,1688	1,4484
n = 12	0,03137	0,1193	0,3357	0,5453	0,7880	1,1714	1,4521
n = 15	0,03424	0,1244	0,3412	0,5500	0,7926	1,1773	1,4606
n = 20	0,03807	0,1298	0,3461	0,5547	0,7975	1,1839	1,4698
n = 30	0,04354	0,1351	0,3509	0,5605	0,8036	1,1916	1,4801
n > 30	0,07089 $-0,15/\sqrt{n}$	0,1601 $-0,14/\sqrt{n}$	0,3793 $-0,15/\sqrt{n}$	0,5887 $-0,15/\sqrt{n}$	0,8326 $-0,16/\sqrt{n}$	1,2239 $-0,17/\sqrt{n}$	1,5174 $-0,20/\sqrt{n}$

Se presenta otra vez el problema de la elección del número de observaciones *n*. A pesar de que elegir valores de *n* grandes hará que ambas distribuciones se parezcan más, tenderá a promediar (encubrir) comportamientos no aleatorios locales. Por otra parte, el hecho de que se tengan que almacenar y ordenar las observaciones en orden ascendente, hace preferible la elección de valores de *N* comparativamente pequeños. Un buen compromiso sería tomar *n* igual, por ejemplo, a 1.000, y realizar el experimento en diferentes partes de la secuencia obteniendo $K_{1000}^+(1), K_{1000}^+(2), \ldots, K_{1000}^+(r)$, y aplicar entonces otro test KS a los resultados obtenidos. Este método consiste en usar diversos tests con valores de *N* de tamaños moderados y combinar sus resultados con otro test de KS que tiende a detectar comportamientos no aleatorios tanto locales como globales.

El test de KS se puede usar en combinación con el test chi-cuadrado (χ^2). Esta forma de proceder ofrece mayor calidad de decisión sobre varios resultados obtenidos al aplicar el test chi-cuadrado que hacer tres de estos tests y mirar independientemente cuántos de ellos resultan sospechosos. Suponer para ello que se realizan 10 tests chi-cuadrado distintos sobre diferentes partes de una secuencia aleatoria y que se obtienen los valores χ_1^2, χ_2^2,..., χ_{10}^2. No es una buena filosofía contar simplemente cuántos resultados son sospechosos (por arriba o por abajo, aunque este procedimiento trabaja en casos extremos y por tanto, valores muy grandes o muy pequeños pueden significar que la secuencia tiene no aleatoriedades locales muy acentuadas). Un método más útil sería dibujar la distribución empírica de estos 10 valores, y compararla con la distribución correcta (esta

distribución se puede encontrar en la tabla 5.3). Esto da una conclusión acerca de los resultados del test chi-cuadrado más realista, y las estadísticas K_{10}^{+} y K_{10}^{-} se pueden determinar como una indicación de éxito o fracaso.

Se puede concluir estableciendo un procedimiento básico a seguir a la hora de aplicar la mayoría de los tests estadísticos que se verán a continuación:

En primer lugar se deben realizar *N* tests chi-cuadrado distintos, cada uno, de *n* observaciones próximo al mínimo requerido sobre la secuencia, y aplicar sobre los resultados obtenidos un test de Kolmogorov-Smirnov. Al hacerlo, estaremos evaluando el conjunto de la secuencia y no sólo una de las subsecuencias testeadas en cualquiera de los *N* tests. Una vez hecho esto, se puede emplear el mismo criterio de valoración que para el test chi-cuadrado, con el nivel de significación del 5% que es ampliamente recomendado. Si $5\% < P_{K_n} < 95\%$ se podrá aceptar a la secuencia como aleatoria, ya que no habrán razones fundadas para pensar lo contrario. Sin embargo, si $P_{K_n} < 5\%$ o $P_{K_n} > 95\%$, se podrá considerar a la secuencia como defectuosa, y rechazar, por tanto, el generador que la produjo.

5.2.3 Tests estadísticos empíricos

En la práctica será necesario testear cuidadosamente cualquier secuencia pseudoaleatoria que generemos para ser utilizada en una determinada aplicación. Si bien existen gran número de tests que evalúan la aleatoriedad de las secuencias, sólo se van a ver los que parecen tener mayor utilidad. Hay que notar que el hecho que una secuencia haya pasado con éxito *k* tests estadísticos cualesquiera T_1, T_2,..., T_k, no nos garantiza en absoluto que no vaya a ser rechazada al aplicarle un nuevo test T_{k+1}. Sin embargo, cada uno de ellos nos dará más y más confianza sobre la aleatoriedad de la secuencia. En la práctica se suelen aplicar una docena de estos tests y, si la secuencia los pasa satisfactoriamente, se la considera aleatoria.

Un test estadístico *T* [CAR89] es una función $T : X^N \rightarrow \{\text{acepta, rechaza}\}$, que divide el conjunto de secuencias binarias de longitud *N* (X^N) en un conjunto más pequeño:

$$S_T = \{ x^N : T(x^N) = \text{rechazado} \} \subseteq X^N$$

o de *malas* secuencias y el resto en un conjunto de *buenas* secuencias. Los dos parámetros principales de un test estadístico son la longitud de la muestra de la secuencia *N* y la tasa de rechazo $\rho = |S_T| / 2^N$ (también llamada nivel de significancia), que es la probabilidad de que la secuencia estudiada sea rechazada.

Un defecto estadístico de un generador de bits aleatorios sólo podrá detectarse con una cierta probabilidad, que dependerá de la seriedad del defecto y de la longitud *N* de la muestra de la secuencia. Existe un compromiso entre la probabilidad de detección y la probabilidad de falsa alarma. En un test práctico, la tasa de rechazo ρ debe ser pequeña, por ejemplo $\rho \cong 0,001 \ldots 0,01$. Por razones prácticas, un test estadístico aplicado a una muestra suficientemente grande de la secuencia no se puede implementar listando el conjunto de malas secuencias. Es mejor especificar una función eficiente y computable f_T, que asocie las secuencias binarias de longitud *N* a números reales *R*:

$$f_T : B^N \rightarrow R : x^N \rightarrow f_T(x^N)$$

Una vez establecida esta función f_T, se debe determinar la función de distribución de probabilidad de la variable real aleatoria $f_T(R^N)$, donde R^N denota una secuencia de N variables binarias estadísticamente independientes y simétricamente distribuídas. Además, se han de especificar también unas cotas superior (t_1) e inferior (t_2) que nos permitan aceptar o rechazar la secuencia, de forma que:

$$Pr[f_T(R^N) \le t_1] + Pr[f_T(R^N) \ge t_2] = \rho$$

Usualmente $Pr[f_T(R^N) \le t_1] \approx Pr[f_T(R^N) \ge t_2] \approx \rho/2$. El conjunto S_T de secuencias malas con cardinalidad $|S_T| = \rho 2^N$ se puede definir entonces como:

$$S_T = \{x^N \in B^N : f_T(x^N) \le t_1 \text{ ó } f_T(x^N) \ge t_2\}$$

Normalmente, f_T se elige de tal forma que $f_T(R^N)$ esté distribuída (aproximadamente) de acuerdo con alguna función de distribución bien conocida. La que se usa más a menudo es la distribución chi-cuadrado (χ^2) con ν grados de libertad, donde ν es algún entero positivo. Esta distribución se obtiene al sumar los cuadrados de ν variables aleatorias independientes y normalmente distribuídas con media 0 y varianza 1. Además, es fácil encontrar tablas de esta distribución, lo que simplifica mucho el cálculo de t_1 y t_2 una vez se ha especificado la tasa de rechazo ρ y la longitud de la muestra que se va a estudiar (N).

Es interesante notar que, incluso aunque la secuencia estudiada tenga las propiedades estadísticas deseadas (por ejemplo, su secuencia de salida sea indistinguible de una secuencia verdaderamente aleatoria) y, por tanto, haya pasado el test, no podemos garantizar que otra secuencia producida por este generador u otra porción de la misma secuencia pasará de nuevo el test, ya que un ρ% de todas las secuencias fallarán. Por tanto, si una única secuencia de salida del generador falla al pasar el test, esto no implica una debilidad de éste. En la mayoría de los casos, lo mejor será comprobar un número elevado de muestras y analizar los resultados antes de dar ninguna conclusión.

Se pueden distinguir dos tipos de tests: los tests empíricos y los teóricos. En los primeros, un ordenador manipula grupos de números de la secuencia y evalúa ciertas estadísticas, mientras que en los teóricos se establecen ciertas características de la secuencia mediante el uso de métodos numéricos teóricos basados en la regla de recurrencia que se usa para formar la secuencia. A continuación se describen algunos de los tests empíricos que se pueden pasar sobre las secuencias obtenidas de los generadores pseudoaleatorios.

A) Test de frecuencia o de equidistribución

El primer requisito que debe cumplir una secuencia es que el número de ceros y unos esté equidistribuído (es decir, que presenten una distribución uniforme). Si se dispone de una muestra de N bits $x^N = x_1,..., x_N$, sobre la que se quiere estudiar si cumple esta propiedad, se puede definir la función de distribución $f_{T_f}(x^N)$ como [KNU67]:

$$f_{T_f}\left(x^N\right) = \frac{2}{\sqrt{N}}\left(\sum_{i=1}^{N} x_i - \frac{N}{2}\right)$$

El número de unos en una secuencia aleatoria de N bits $r^N = r_1,..., r_N$ estará distribuído aproximadamente según una distribución normal de media $N/2$ y varianza $N/4$, ya que $E[r_i]=1/2$ y $Var[r_i]=1/4$ para $1 \leq i \leq N$.

Esto indica que la función de distribución de probabilidad $f_{T_f}(x^N)$ se aproximará más y más a una función de distribución normal de media 0 y varianza 1 a medida que el número N de bits de la muestra estudiada sea suficientemente grande.

Una forma práctica de pasar este test consiste en aplicar un test chi-cuadrado sobre la muestra que queremos estudiar, teniendo en cuenta que se tendrán dos categorías ($k = 2$), ya que la observación realizada será bien un cero o bien un uno, y por tanto tendrá $\nu = k-1 = 1$ grados de libertad.

Para evitar resultados falseados debidos a posibles no aleatoriedades locales en la que pudiera haber coincidido la muestra elegida, es conveniente aplicar el test chi-cuadrado un número determinado M de veces (por ejemplo, 10 es un número recomendable en la práctica) sobre diferentes partes de la secuencia, y sobre estos resultados de los tests chi-cuadrado realizados aplicar el test de Kolmogorov-Smirnov, de forma que el resultado sea una indicación de si la secuencia pasa o no, el test. Las estadísticas del test de Kolmogorov-Smirnov se pueden encontrar en la tabla 5.3 de este capítulo. Por tanto, podemos definir un algoritmo para este test como:

Paso 1	Generar una secuencia de M conjuntos consecutivos, cada uno de N números aleatorios.
Paso 2	Hacer una partición del rango de números en k intervalos.
Paso 3	Tabular la frecuencia dentro de cada intervalo para cada uno de los M grupos.
Paso 4	Comparar los resultado de los M grupos entre ellos y con los valores esperados (distribución uniforme continua) usando el test chi-cuadrado con $\nu = 1$ grados de libertad.

Unos valores interesantes para pasar este test son $M = 10$ y $N = 100$.

B) Tests sucesivos

Estos test miden el grado de aleatoriedad entre números consecutivos de la secuencia pseudoaleatoria. Es decir, además de que los unos y los ceros estén uniformemente distribuídos, tal como medía el test de frecuencia, también interesará que las parejas ($L = 2$), trios ($L = 3$), cuartetos ($L = 4$), etc., de bits estén uniformemente distribuídos aplicando el test T_S con parámetro L. Para ello, se debe dividir la secuencia de muestra de N bits x^N en N/L bloques consecutivos de longitud L. Se deberá entonces ir mirando dentro de qué categoria i van cayendo cada uno de los sucesivos bloques, generando la tabla $n_i(x^N)$ con el número de occurencias de las diferentes categorías.

Puesto que cada bloque consta de L bits, evidentemente las categorías estarán en el rango $0 \leq i \leq 2^L$-1. A partir de esta tabla se puede obtener la función de distribución de probabilidad f_{T_s} como [GRAYB]:

$$f_{T_s}\left(s^N\right) = \frac{L2^L}{N} \sum_{i=0}^{2^L-1} \left(n_i\left(s^N\right) - \frac{N}{L2^L} \right)^2$$

Cuando el valor de la muestra N sea suficientemente elevado, esta función de distribución deberá aproximarse a una función chi-cuadrado con 2^L-1 grados de libertad.

Veamos el caso de distribución de parejas y tríos de bits de la secuencia.

B1) Test de parejas

Este test [KNU67] mira si las parejas $(x_{2j}, x_{2j+1}) = (q,r)$ están uniformemente distribuídas. Para ello se debe dividir la secuencia que se quiere estudiar en bloques de 2 bits ($L = 2$), y se debe mirar el número de veces que ocurre cada una de las categorías posibles ($k = 4$ en este caso : 00, 01, 10, 11), y aplicar sobre ellos un test chi-cuadrado con $\nu = k - 1 = 3$ grados de libertad. La probabilidad de que un suceso caiga dentro de cada categoría será, pues, de 1/4. El número de observaciones que se deben realizar se debe coger de forma que $n > 5 \times 4$.

La forma adecuada de realizar este test consistirá en aplicar un número determinado de tests chi-cuadrado (por ejemplo 10) sobre diversas partes de la secuencia, y sobre estos resultados aplicar un test de Kolmogorov-Smirnov. Se puede definir un procedimiento general para llevar a cabo este test de la siguiente forma:

Paso 1	Generar una secuencia de M grupos consecutivos cada uno con N números pseudoaleatorios.
Paso 2	Hacer una partición del rango de números en k intervalos.
Paso 3	Para cada grupo, construir una matriz de tamaño $k \times k$ e inicializarla con ceros.
Paso 4	Examinar la secuencia de izquierda a derecha considerando pares de números (no considerando cada número dos veces). Si el miembro izquierdo del par está en el intervalo i mientras que el de la derecha está en el intervalo j, incrementar el elemento (i,j) de la matriz en 1.
Paso 5	Cuando se haya rellenado la matriz para cada grupo, comparar los resultados obtenidos con cada uno de los M grupos entre ellos y con los valores esperados usando un test chi-cuadrado (cada par debe ser equiprobable).

Unos valores interesantes para pasar este test son $M = 10$ y $N = 100$. Este test se puede extender de forma que permita examinar la equidistribución de grupos de 3, 4,... bits.

El número de bits necesarios para obtener un test válido en estos casos será, evidentemente, mayor.

B2) Test de tríos

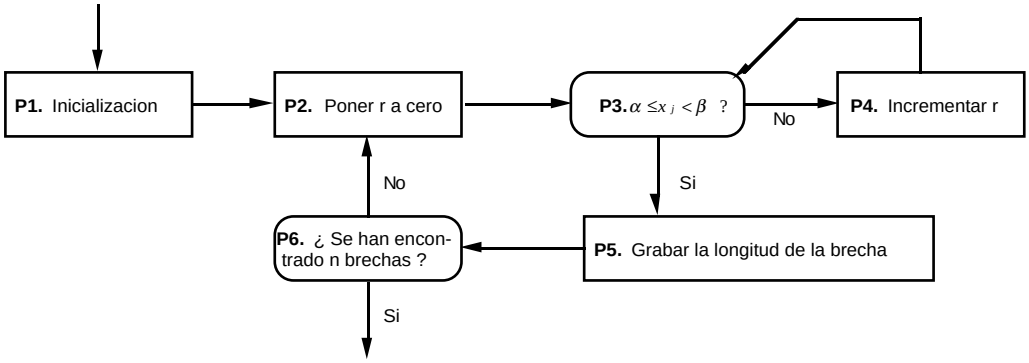
Este test [KNU67] comprueba que las ternas de bits (L = 3) estén equidistríbuidas. En este caso, el número de categorías será k = 8 (ya que con 3 bits se pueden obtener 8 combinaciones), por lo que se debe aplicar un test chi-cuadrado con ν = 7 grados de libertad, y la probabilidad de que una observación caiga dentro de cada categoría será de 1/8. Una vez más, se deberá realizar un número determinado de tests chi-cuadrado (por ejemplo 10) sobre diversas partes de la secuencia, y sobre estos resultados aplicar un test de Kolmogorov-Smirnov para obtener una indicación fiable de si la secuencia pasa o no el test.

Los tests sucesivos se pueden generalizar para cuartetos (L = 4), quíntuplas (L = 5), sextuplas (L = 6), etc, aplicando entonces tests chi-cuadrado con $k = d^L$ categorías con probabilidad $1/d^L$ para cada categoría, con d como base de los números que constituyen la secuencia que se estudia (d = 2 para el caso binario).

C) Test de brechas

Este test [KNU67] se aplica sobre la secuencia $\{x_n\} = x_0, x_1, x_2, ...$ para examinar la longitud de las brechas entre ocurrencias de x_j en un cierto rango. Si α y β son dos números reales tales que $0 \leq \alpha < \beta \leq 1$, queremos considerar las longitudes de subsecuencias consecutivas $x_j, x_{j+1},..., x_{j+r}$, en las que x_{j+r} caiga entre α y β, y las otras x´s caigan fuera (esta subsecuencia de $r + 1$ números representa una brecha de longitud r).

El siguiente algoritmo, aplicado sobre la secuencia $\{x_n\}$, cuenta el número de brechas de longitudes 0, 1,..., t - 1 y el número de brechas de longitud $\geq t$, hasta que se hayan tabulado n brechas.



Este algoritmo es similar a los que se usarán en el test colector de cupones y el test de subsecuencias que se verán posteriormente. Para ello se utiliza un vector, CONT[r] con $0 \leq r \leq t$, que se usará para llevar el cuenteo de las brechas con longitudes entre 0 y t.

Paso 1	[Inicializar] Hacer $j \leftarrow -1$, $s \leftarrow 0$, y CONT[r] $\leftarrow 0$ desde $0 \leq r \leq t$.
Paso 2	[Poner r a cero] $r \leftarrow 0$.
Paso 3	[$\alpha \leq x_j \leq \beta$?] Incrementar j en 1. Si $x_j \geq \alpha$ y $x_j < \beta$, ir al paso 5.
Paso 4	[Incrementar r] Incrementar r en uno y volver al paso 3.
Paso 5	[Grabar la longitud de la brecha] (Se ha encontrado una brecha de longitud r). Si $r \geq t$, incrementar CONT[t] en uno. En otro caso, incrementar CONT[r] en uno.
Paso 6	[¿Se han encontrado n brechas?] Incrementar s por uno. Si $s < n$, volver al paso 2.

Después de realizar este algoritmo, se puede aplicar un test chi-cuadrado a los $t = k+1$ valores CONT[0], CONT[1], ..., CONT[t], usando para ello las siguientes probabilidades: $p_0 = p$, $p_1 = p(1-p)$, $p_2 = p(1-p)^2$,..., $p_{t-1} = p(1-p)^{t-1}$, $p_t = p(1-p)^t$, donde $p = \beta - \alpha$ y es la probabilidad de que $\alpha \leq x_j < \beta$. Una regla práctica consiste en elegir los valores de n y t de forma que cada uno de los valores de COUNT[r] sea igual o mayor que 5.

El test de brechas de aplica a menudo con $\alpha = 0$ o $\beta = 1$ para facilitar el test en el paso 3. Los casos especiales en los que (α, β)= (0,1/2) o (1/2,1) dan lugar a los tests llamados normalmente carreras por debajo de la media (*runs below the mean*) y carreras por encima de la media (*runs above the mean*) respectivamente, cuyo listado en lenguaje C se puede ver en el programa que se adjunta con el libro.

D) Test de poker (test de particiones)

Este test estadístico [GRAYB][KNU67] examina grupos sucesivos de 5 números de la secuencia pseudoaleatoria. Para ello, considera n grupos de cinco enteros sucesivos, (y_{5j}, y_{5j+1},..., y_{5j+4}), $0 \leq j < n$, y observa en cuál de las siguientes categorías cae cada quíntupla:

Todas diferentes:	abcde
Una pareja:	aabcd
Doble pareja:	aabbc
Trío:	aaabc
Full:	aaabb
Poker:	aaaab
Repoker:	aaaaa

Entonces se aplica un test chi-cuadrado basado en el número de quíntuplas que hayan caído dentro de cada categoría. Sin embargo, existe una versión de este test más simple que facilita su

programación, fundamentada en un compromiso basado en contar simplemente el número de elementos distintos en cada quíntupla. Esto implica que se considerarán sólo cinco categorías:

5 diferentes:	todas diferentes.
4 diferentes:	una pareja.
3 diferentes:	doble pareja o trío.
2 diferentes:	full o poker.
1 diferente:	repoker.

En general, para realizarlo podemos considerar n grupos de k (5 en nuestro caso) números sucesivos y podemos contar el número de k-tuplas con r (5 categorías en nuestro caso) valores diferentes. Entonces se aplica un test chi-cuadrado usando las probabilidades (habrán r diferentes):

$$p_r = \frac{d(d-1)\cdots(d-r+1)}{d^k}\left\{ {k \atop r} \right\}$$

donde $\left\{ {k \atop r} \right\}$ representa los números de Stirling de segunda clase [KNUTH], que indican el número de formas de hacer particiones de r subconjuntos disjuntos no vacíos sobre un conjunto de k elementos. Por ejemplo, el conjunto {1, 2, 3, 4} se puede partir en dos subconjuntos de $\left\{ {4 \atop 2} \right\} = 7$ formas distintas: {1, 2, 3}{4}; {1, 2, 4}{3}; {1, 3, 4}{2}; {2, 3, 4}{1}; {1, 2}{3, 4}; {1, 3}{2, 4} y {1, 4}{2, 3}. Veamos cómo calcular estos números. Los números de Stirling son los coeficientes implicados en la transformación entre un polinomio expresado en potencias de k y su representación en forma de coeficientes binomiales. Actualmente no existe un acuerdo global sobre la notación de estos números. La notación usada aquí, considera todos los números de Stirling como no negativos, lo que hace más sencillo recordar su analogía con los coeficientes binomiales. Los números de Stirling de primera especie $\left[{k \atop r} \right]$ se usan para convertir coeficientes binomiales en potencias:

$$n!\binom{x}{n} = x(x-1)\cdots(x-n+1) = \left[{n \atop n} \right]x^n - \left[{n \atop n-1} \right]x^{n-1} + \cdots + (-1)^n\left[{n \atop 0} \right] = \sum_k (-1)^{n-k}\left[{n \atop k} \right]x^k$$

donde:
$$\binom{n}{k} = \frac{n(n-1)\cdots(n-k+1)}{k(k-1)\cdots(1)}$$

mientras que los de segunda especie, y que son los que se calculan para realizar los tests estadísticos que se estudian, se usan para convertir potencias en coeficientes binomiales:

$$x^n = \left\{ {n \atop n} \right\}\binom{x}{n}n! + \cdots + \left\{ {n \atop 1} \right\}\binom{x}{1}1! + \left\{ {n \atop 0} \right\}\binom{x}{0}0! = \sum_k \left\{ {n \atop k} \right\}\binom{x}{k}k!$$

La tabla 5.4 muestra el triángulo de Stirling para la segunda especie, que en cierta forma es análogo al triángulo de Pascal, y permite el cálculo de los primeros valores de los números de Stirling de segunda especie. Para calcular otros valores se puede usar la siguiente fórmula de recurrencia:

$$\left\{ {n \atop m} \right\} = m\left\{ {n-1 \atop m} \right\} + \left\{ {n-1 \atop m-1} \right\} \qquad \text{si } n > 0.$$

además de considerar las siguientes relaciones:

$$\left\{ {n \atop 0} \right\} = 0, \qquad \left\{ {n \atop 1} \right\} = 1$$

y

$$\left\{ {n \atop 2} \right\} = 2^{n-1} - 1 \qquad \text{si } n > 0.$$

Tabla 5.4 Triángulo de Stirling

	$\left\{ {n \atop 0} \right\}$	$\left\{ {n \atop 1} \right\}$	$\left\{ {n \atop 2} \right\}$	$\left\{ {n \atop 3} \right\}$	$\left\{ {n \atop 4} \right\}$	$\left\{ {n \atop 5} \right\}$	$\left\{ {n \atop 6} \right\}$	$\left\{ {n \atop 7} \right\}$	$\left\{ {n \atop 8} \right\}$
$n = 0$	1	0	0	0	0	0	0	0	0
$n = 1$	0	1	0	0	0	0	0	0	0
$n = 2$	0	1	1	0	0	0	0	0	0
$n = 3$	0	1	3	1	0	0	0	0	0
$n = 4$	0	1	7	6	1	0	0	0	0
$n = 5$	0	1	15	25	10	1	0	0	0
$n = 6$	0	1	31	90	65	15	1	0	0
$n = 7$	0	1	63	301	350	140	21	1	0
$n = 8$	0	1	127	966	1701	1050	266	28	1

E) Test colector de cupones

El nombre de este test [KNU67] se debe a que recuerda a una persona recogiendo c tipos de cupones diferentes que estén aleatoriamente distribuídos en las cajas de cereales del desayuno, de forma que debe seguir comiendo más cereales hasta que tenga un cupón de cada tipo. Este test está relacionado con el test de poker, de la misma forma que el test de brechas está relacionado con el test de frecuencia.

A diferencia del anterior, que usaba una secuencia de números reales, este test usa una secuencia de enteros $y_0, y_1,...$, en el que se observan las longitudes de los segmentos $y_{j+1}, y_{j+2},..., y_{j+r}$ requeridos para conseguir un conjunto completo de enteros del conjunto que va 0 a c - 1.

Veamos el algoritmo que realiza esto: dada una secuencia de enteros $y_0, y_1,...$, con $0 \leq y_j < c$, este algoritmo cuenta las longitudes de n segmentos consecutivos que recogen el conjunto completo de enteros. Al finalizar el algoritmo, el vector CONT[r] contiene el número de segmentos de longitud r, para $0 \leq r < t$, y CONT[t] contendrá el número de segmentos con longitud $\geq t$.

Paso 1	[Inicializar] Poner $j \leftarrow -1$, $s \leftarrow 0$ y CONT[r] $\leftarrow 0$ para $0 \leq r \leq t$.
Paso 2	[Poner q y r a cero] Poner $q \leftarrow r \leftarrow 0$, y poner OCURS[$k$] $\leftarrow 0$, para $0 \leq k <$ c.
Paso 3	[Próxima observación] Incrementar r y j en uno. Si OCURS[Y_j] $\neq 0$, repetir este paso.
Paso 4	[¿Está el conjunto completo?] Poner OCURS[Y_j]$\leftarrow$1, $q \leftarrow q + 1$. (La subsecuencia observada contiene q valores distintos; si $q = c$, tendremos un conjunto completo). Si $q < c$, volver al paso 3.
Paso 5	[Grabar la longitud] Si $r \geq t$, incrementar CONT[t] en uno. En cualquier otro caso, incrementar CONT[r] en uno.
Paso 6	[¿Se han encontrado n segmentos?] Incrementar s en uno. Si $s < n$,volver al paso 2.

El vector OCURS mantiene el conteo de los segmentos (número de ocurrencias). Una vez el algoritmo anterior ha contado n segmentos se debe aplicar un test chi-cuadrado con $k = t - c + 1$ grados de libertad sobre CONT[c], CONT[$c + 1$],..., CONT[t], usando las probabilidades:

$$p_r = \frac{c!}{c^r} \left\{ \begin{matrix} r-1 \\ c-1 \end{matrix} \right\} \qquad c \leq r < t$$

Para obtener estas probabilidades, basta observar que si q_r indica la probabilidad de que una subsecuencia de longitud r esté incompleta (es decir, que tenemos una r-tupla de elementos que no poseen todos y cada uno de los c valores). En este caso, las probabilidades anteriores se pueden calcular con las relaciones: $p_r = q_{r-1} - q_r$ para $d \leq r < t$, y $p_t = q_t - 1$.

F) Test de distancia

Este test [GRAYB] considera a pares sucesivos de una secuencia de números pseudoaleatorios entre 0 y 1 como coordenadas de puntos dentro del cuadrado unidad (cuadrado de lado 1). Si la secuencia de que se dispone es $\{x_n\} = x_1, x_2, x_3,..., x_n$, los puntos (x_1,x_2), (x_3,x_4),..., (x_{n-1},x_n) se pueden representar en un plano con ejes x-y. Se puede calcular entonces la distancia euclídea D^2 entre todos los puntos, obtenida a partir de la secuencia $\{x_n\}$. Si estos puntos estuviesen aleatoriamente distribuídos dentro del cuadrado unidad (lo que ocurriría si la secuencia fuese verdaderamente aleatoria) la probabilidad de que los valores observados de D^2 fuesen menores o iguales que un valor determinado

k vendría dada por la función de distribución:

$$F(k)=\begin{cases}\pi k-\dfrac{8}{3}k^{3/2}+\dfrac{k^2}{2} & \text{para} \quad k\le 1,0\\[2ex] \dfrac{1}{3}+(\pi-2)k+4(k-1)^{1/2}+\dfrac{8}{3}(k-1)^{3/2}-\dfrac{k^2}{2}-4k\cdot arc\sec\sqrt{k} & \text{para } 1,0\le k\le 2,0\end{cases}$$

Según esto, F(0,0) = 0, F(0,25) = 0,483, F(0,5) = 0,753, F(0,75) = 0,905, F(1,0) = 0,975, F(1,5) = 0,999 y F(2,0) = 1,0. Mediante el uso de esta distribución, se pueden calcular los valores teóricos de las frecuencias para los valores de D^2 en cualquier intervalo deseado. Esto servirá para realizar un test basado en la distancia euclídea, comparando los valores de frecuencias observados con los valores teóricos obtenidos a partir de la distribución anterior y aplicando un test chi-cuadrado.

G) Test de permutaciones

Este test [KNU67], que es una generalización del test de poker, divide la secuencia que se quiere comprobar en *n* grupos de *t* elementos cada uno, es decir, (x_{jt}, x_{jt+1},..., x_{jt+t-1}), $0 \le j < n$. Los elementos de cada grupo pueden tener *t*! reordenamientos posibles. Lo que se hace es contar el número de veces que aparece cada reordenamiento y se aplica un test chi-cuadrado con $k = t!$ grados de libertad y con una probabilidad de $1/t!$ para cada reordenamiento. Por ejemplo, si $t = 3$ tendremos 6 categorías, determinadas según $x_{3j} < x_{3j+1} < x_{3j+2}$, o $x_{3j} < x_{3j+2} < x_{3j+1}$,..., $x_{3j+2} < x_{3j+1} < x_{3j}$.

En este test se asume que no puede haber un número de la secuencia (*x*´s) seguido igual, por lo que conviene eliminar las repeticiones de números consecutivos que aparecen en la secuencia antes de pasar el test. Para realizar este test se usa el siguiente algoritmo: dado un conjunto de elementos distintos (x_1, x_2,..., x_t), se computa un entero f(x_1, x_2,..., x_t), de forma que $0 \le f(x_1, x_2,..., x_t) < t!$, y f($x_1$,..., x_t) = f(z_1, ..., z_t) si, y sólo si, (x_1, x_2,..., x_t) y (z_1,...,z_t) presentan el mismo ordenamiento. Para ello se debe usar una tabla auxiliar C[1],...,C[*t*]. Veamos cómo realizar esto de forma algorítmica:

Paso 1 [Inicializar] Inicializar $r \leftarrow t$.

Paso 2 [Encontrar el máximo] Encontrar el máximo de $\{x_1,..., x_r\}$, y suponer que x_s es el máximo. Poner C[*r*] $\leftarrow$ *s* - 1.

Paso 3 [Intercambiar] Intercambiar $x_r \leftrightarrow x_s$.

Paso 4 [Decrementar *r*] Decrementar *r* en uno. Si $r > 0$, volver al paso 2.

Paso 5 [Computar f] El valor de la función deseada viene dado por la fórmula:

f = C[*t*]+tC[*t*-1]+*t*(*t*-1)C[*t*-2]+...+*t*!C[1] = (...((C[1]×2+C[2])×3+C[3])+...+C[*t*- 1])×t+C[*t*].

Por la estructura de este algoritmo, se observa que $0 \leq C[r] < r$, por lo que C[1] será siempre cero. Además, $0 \leq f < t!$. Cuando el algoritmo finalice, $(x_1,..., x_t)$ se habrá ordenado en sentido asecendente.

H) Test de subsecuencias

Este test se emplea para comprobar la aleatoriedad de la oscilación de los números que componen la secuencia. En una secuencia se pueden estudiar las subsecuencias ascendentes y las subsecuencias descendentes. Esto significa examinar las longitudes de subsecuencias monótonas de la secuencia original (por ejemplo, segmentos en los que números sucesivos se van incrementando o decrementando).

Como ejemplo de definición de una subsecuencia, consideremos una secuencia de 10 números 1298536704, donde ponemos una línea vertical a la izquierda y a la derecha entre dos números x_j y x_{j+1}, siempre que $x_j > x_{j+1}$: / 1 2 9 / 8 / 5 / 3 6 7 / 0 4 /. Esto indica las subsecuencias ascendentes, y sale una subsecuencia de longitud 3, seguida de dos subsecuencias de longitud 1, seguida de otra subsecuencia de longitud 3, y finalmente una subsecuencia de longitud 2.

A diferencia del test de brechas y del test colector de cupones, que son parecidos a este test en muchos sentidos, aquí no se debe aplicar un test chi-cuadrado sobre los datos obtenidos, ya que las subsecuencias adyacentes no son independientes. Se observa que una subsecuencia larga tiende a ir seguida por una corta e igualmente a la inversa. Esta falta de independencia es suficiente para invalidar la validez de un test chi-cuadrado.

Veamos, pues, cómo se realiza este test: Sea $<x_n> = x_0, x_1,..., x_{n-1}$ una secuencia que se quiere comprobar de n números distintos. El algoritmo para realizar el test de subsecuencias ascendentes se puede ver en el listado del programa correspondiente que se adjunta con el libro, donde se puede ver qué determina las longitudes de todas las subsecuencias ascendentes en la secuencia, de forma que cuando el algoritmo termina, el vector CONT[r] contendrá el número de subsecuencias de longitud r, para $1 \leq r \leq 5$, y CONT[6] será el número de subsecuencias de longitud 6 o más. Para el test de subsecuencias descendentes se debe realizar un algoritmo similar, pero ahora debe determinar las longitudes de todas las subsecuencias descendentes.

Una vez obtenidos estos datos debe determinarse [KNU67] la siguiente estadística:

$$V = \frac{1}{n} \sum_{1 \leq i,j \leq 6} (CONT[i] - nb_i)(CONT[j] - nb_j) a_{ij}$$

donde los coeficientes a_{ij} y b_j son los siguientes:

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} \end{vmatrix} = \begin{vmatrix} 4529.4 & 9044.9 & 13568 & 18091 & 22615 & 27892 \\ 9044.9 & 18097 & 27139 & 36187 & 45234 & 55789 \\ 13568 & 27139 & 40721 & 54281 & 67852 & 83685 \\ 18091 & 36187 & 54281 & 72414 & 90470 & 111580 \\ 22615 & 45234 & 67852 & 90470 & 113262 & 139476 \\ 27892 & 55789 & 83685 & 111580 & 139476 & 172860 \end{vmatrix}$$

$$(b_1 \quad b_2 \quad b_3 \quad b_4 \quad b_5 \quad b_6) = \left(\frac{1}{6} \quad \frac{5}{24} \quad \frac{11}{120} \quad \frac{19}{720} \quad \frac{29}{5040} \quad \frac{1}{840}\right)$$

Estos valores, que son los que se utilizan para implementar el algoritmo, son sólo aproximados. Los valores exactos pueden determinarse a partir de las expresiones que se pueden encontrar en [KNU67]. La estadística *V* obtenida antes, debe tener una distribución chi-cuadrado con 5 (y no 6 como podría parecer) grados de libertad cuando *n* es grande. Podemos concluir este test proponiendo el siguiente algoritmo para desarrollarlo:

Paso 1	Generar *N* números pseudoaleatorios (por ejemplo 10.000).
Paso 2	Construir una secuencia binaria de forma que para cada dos números consecutivos x_j y x_{j+1} de la secuencia, el bit *j*-ésimo sea 0 si $x_j < x_{j+1}$ y 1 en caso contrario.
Paso 3	Tabular la frecuencia de ocurrencia de las subsecuencias (tanto de ceros como de unos) para cada longitud (2, 3, 4, 5,...).
Paso 4	Comparar las frecuencias tabuladas con los valores esperados empleando la estadística anterior.

I) Test de correlación sucesiva

Según [KNU67], para pasar este test sobre una secuencia (o subsecuencia) de *n* elementos x_0, x_1,..., x_n, se debe calcular la siguiente estadística:

$$C = \frac{n(x_0x_1 + x_1x_2 + \cdots + x_{n-2}x_{n-1} + x_{n-1}x_0) - (x_0 + x_1 + \cdots + x_{n-1})^2}{n(x_0^2 + x_1^2 + \cdots + x_{n-1}^2) - (x_0 + x_1 + \cdots + x_{n-1})^2} \tag{8}$$

A este coeficiente se le suele denominar de correlación sucesiva, y es una medida de cuánto depende x_{j+1} de x_j. Los coeficientes de correlación aparecen muy frecuentemente en estadística. Si tenemos dos secuencias de *n* números x_0, x_1,..., x_{n-1} y z_0, z_1,..., z_{n-1}, el coeficiente de correlación entre ellos se define como:

$$C = \frac{n\sum(x_jz_j) - (\sum x_j)(\sum z_j)}{\sqrt{\left(n\sum x_j^2 - (\sum x_j)^2\right)\left(n\sum z_j^2 - (\sum z_j)^2\right)}}$$

donde todos los sumatorios se deben realizar en el rango $0 \le j < n$. La ecuación (8) es el caso especial en que $z_j = x_{(j+1) \bmod n}$. Un coeficiente de correlación siempre cae entre +1 y -1. Cuando es cero o muy

pequeño, indica que las cantidades x_j y z_j son mutuamente independientes, pero cuando el coeficiente de correlación es ±1, indica una total dependencia lineal.

En este caso, $z_j = m \pm ax_j$ para algunas constantes a y m. Por tanto, es deseable obtener un valor de C cercano a cero. Si x_0x_1 no es completamente independiente de x_1x_2, el coeficiente de correlación serie no será exactamente cero. Un buen valor de C será aquel que esté entre $\mu_n - 2\sigma_n$ y $\mu_n + 2\sigma_n$, donde:

$$\mu_n = \frac{-1}{(n-1)} \qquad \sigma_n = \frac{1}{n-1}\sqrt{\frac{n(n-3)}{n+1}} \qquad n > 2$$

y se esperará que C caiga entre estos dos valores el 95% de las veces.

J) Test de transiciones

Este test se utiliza para asegurar que las probabilidades de transición son razonables, es decir, la probabilidad de que dos bits sean iguales o diferentes sean más o menos la misma. Si ocurre así, se puede tener confianza de que cada bit es independiente de su antecesor.

Supongamos que la transición 01 ocurre n_{01} veces, la 10 ocurre n_{10}, la 00 ocurre n_{00} y la 11 ocurre n_{11}. Se puede ver que:

$$n_{01} + n_{00} = n_0 \text{ (ó } n_0 - 1)$$
$$n_{10} + n_{11} = n_1 \text{ (ó } n_1 - 1)$$
$$n_{10} + n_{01} + n_{00} + n_{11} = n - 1$$

El -1 se debe a que en una secuencia de longitud de m bits sólo hay m - 1 transiciones. Idealmente, se desea que $n_{10} = n_{01} = n_{00} = n_{11} \approx (n- 1)/4$ y en [MOO40] se demuestra que la expresión:

$$S(n) = \frac{4}{n-1}\sum_{i=0}^{1}\sum_{j=0}^{1}(n_{ij})^2 - \frac{2}{n}\sum_{i=0}^{1}(n_i)^2 + 1$$

está aproximadamente distribuída como una función chi-cuadrado con 2 grados de libertad. Una forma práctica de pasar este test consiste en calcular, para 10 fragmentos de la secuencia, los n_{ij} y los n_i, la función $S(n)$ y calcular 10 tests chi-cuadrado con dos grados de libertad, sobre cuyos resultados se aplica un test de Kolmogorov-Smirnov. El resultado dará una indicación de si la secuencia pasa o no el test.

K) Test estadísticos basados en el perfil de complejidad lineal

A continuación se describen algunos tests estadísticos de aleatoriedad basados en el perfil de complejidad lineal de una secuencia binaria. La idea de usar este perfil de las secuencias para construir tests estadísticos fue sugerida simultáneamente por Niederreiter [NIE88] y por Glyn Carter [CAR89]. La complejidad lineal de una secuencia binaria mide la longitud del menor registro de desplazamiento con realimentación linal (LFSR) que puede generar la secuencia y la forma en que crece ésta a medida

que se va estudiando. Este crecimiento determina el perfil de complejidad lineal (LCP) [CAR89][MAS76] que se describirá más detalladamente en el siguiente capítulo . Existen dos formas de complejidad lineal: la complejidad lineal global, que se aplica sobre secuencias binarias infinitas, y la complejidad lineal local, que se aplica sobre secuencias finitas. Este último tipo de secuencias son las que estamos tratando en este libro, y los tests estadísticos descritos aquí hacen, pues, referencia a la complejidad lineal local de secuencias periódicas (finitas).

K1) Test estadístico basado en el número de saltos en el LCP

En primer lugar se estudiará un test estadístico para calcular la aleatoriedad local de secuencias binarias finitas, basado en el número de saltos que aparecen en el perfil de complejidad lineal. Antes de describir este test se verán algunos teoremas que nos serán de utilidad para su comprensión posterior:

TEOREMA 1: Sea $f(n,L,J)$ el número de secuencias de *n* bits con complejidad lineal *L*, y *J* saltos en el perfil de complejidad lineal. Entonces:

$$f(n,0,0) = 1$$

Si $1 \le L \le n$ y $1 \le J \le min(L-1, n-L)+1$ entonces:

$$f(n,L,J) = \binom{min(L-1,n-L)}{J-1} \cdot 2^{min(L,n-L)}$$

Sino:

$$f(n,L,J) = 0$$

TEOREMA 2: Sea J_n una variable aleatoria que represente el número de saltos en el perfil de complejidad lineal de una secuencia binaria aleatoria de *n* bits, y sea $\mu = E(J_n)$ el número medio de saltos en su perfil de complejidad lineal. Entonces:

$$\mu = \begin{cases} \frac{n}{4} + \frac{1}{3} - \frac{1}{3 \cdot 2^n} & si \cdot n \cdot par \\ \frac{n}{4} + \frac{5}{12} - \frac{1}{3 \cdot 2^n} & si \cdot n \cdot impar \end{cases}$$

TEOREMA 3: Sea σ^2 la varianza del número de saltos en el perfil de complejidad lineal de una secuencia binaria aleatoria de *n* bits. Entonces:

$$\sigma^2 = \begin{cases} \frac{n}{8} - \frac{2}{9} + \frac{n}{6 \cdot 2^n} + \frac{1}{3 \cdot 2^n} - \frac{1}{9 \cdot 2^{2n}} & si \cdot n \cdot par \\ \frac{n}{8} - \frac{1}{8} + \frac{n}{6 \cdot 2^n} + \frac{7}{18 \cdot 2^n} - \frac{1}{9 \cdot 2^{2n}} & si \cdot n \cdot impar \end{cases}$$

Veamos un test estadístico para aleatoriedad que chequea el número de saltos en el perfil de complejidad lineal de una secuencia binaria. Este test se basa en que la distribución del número de saltos es aproximadamente normal cuando *n* es grande. Más exactamente, para valores de *n* grandes, la variable $(J_n - \mu)/\sigma$ está distribuída aproximadamente según una distribución estándar normal $N(0,1)$. Este hecho se puede aprovechar para construir un tests estadístico que compruebe que el número de saltos en el LCP de una secuencia de *n* bits no sea significativamente diferente del valor esperado. Por tanto, la secuencia fallará en el test si su LCP tiene demasiados o demasiados pocos saltos (por ejemplo, si hay demasiados saltos de longitud excesiva). El procedimiento del test es el siguiente:

Paso 1	Computar el número de saltos *J* en el LCP de la secuencia.
Paso 2	Computar el valor medio μ de J_n usando el teorema 2.
Paso 3	Computar la varianza σ^2 de J_n usando el teorema 3.
Paso 4	Computar el test estadístico $S = (J_n - \mu)/\sigma$.
Paso 5	La secuencia pasa el test con un nivel de significancia del $\alpha\%$ si, y sólo si, $-C \le S \le C$, donde *C* es el punto superior $\frac{\alpha}{2}\%$ de la distribución normal estándar $N(0,1)$.

En el capítulo siguiente se verá el algoritmo de Massey-Berlekemp que permitirá calcular el perfil de complejidad lineal de una secuencia y que, por tanto, permitirá realizar este test.

Este test es capaz de identificar no aleatoriedades en secuencias binarias que no hubiesen sido detectadas usando otros tests estadísticos.

K2) Test estadístico basado en la distribución de la altura de los saltos del LCP

El que una secuencia pase el test anterior no implica que el LCP de esa secuencia no sea sustancialmente diferente del que se espera para una secuencia verdaderamente aleatoria. Este nuevo test comprueba que la distribución de las alturas de los saltos del LCP de una secuencia binaria no sea significativamente diferente de la esperada para una secuencia verdaderamente aleatoria.

Para una secuencia verdaderamente aleatoria, la altura de cada salto es independiente de las alturas de los otros saltos. Se puede demostrar que las alturas de los saltos en el LCP de una secuencia binaria aleatoria son variables aleatorias independientes e idénticamente distríbuidas siguiendo la distribución G(1/2), donde G(*p*) es la distribución geométrica con parámetro *p*.

Este hecho se puede aprovechar para construir un test de aleatoriedad basado en las alturas de los saltos en el LCP de una secuencia. Este test considera los primeros *F* saltos del perfil, donde *F* es fijo para una secuencia de longitud *n*, y comprueba si las alturas de estos saltos conforman una distribución geométrica con parámetro 1/2. Para ello, se dividen los saltos en *M* clases de acuerdo con sus alturas de forma que, para $k = 1, 2, \ldots, M - 1$, la clase *k*-ésima contenga todos los saltos de altura

k, mientras que la clase *M*-ésima contiene todos los saltos con altura ≥ *M*. *M* será el menor entero, tal que el número esperado de saltos sea menor que 5.

La frecuencia esperada para cada clase es F_{p_k}, donde p_k es la probabilidad de que un salto del LCP esté en la clase *k*. Así, puesto que las alturas de los saltos de la secuencia binaria están distribuídas según una distribución geométrica G(1/2), $p_k = (1/2)^k$ para $k = 1, 2,..., M - 1$, y $p_M = (1/2)^M+(1/2)^{M+1}+... = (1/2)^{M-1}$. Dada una secuencia $x_0, x_1,..., x_{n-1}$, el test procede como sigue:

Paso 1 Calcular el LCP de $x_0, x_1,..., x_{n-1}$.

Paso 2 Calcular $F = \langle \mu - C\sigma \rangle$, que es el menor entero mayor o igual que $\mu - C\sigma$, donde C, μ y σ son las del test anterior de número de saltos. Si la secuencia pasa el test anterior, sabemos que su LCP debe contener al menos *F* saltos.

Paso 3 Definir *M* de forma que sea el menor valor de *k* tal que $F(1/2)^k < 5$. Desde $k = 1, 2,..., M$, calcular f_k, donde:

f_k = número de saltos de altura *k* en los primeros *F* saltos del LCP ($k = 1, 2,..., M - 1$). f_M = número de saltos de altura ≥ *M* en los primeros *F* saltos del LCP.

Paso 4 Computar la estadística:

$$S = \sum_{k=1}^{M} \left[\left(f_k - Fp_k \right)^2 \Big/ Fp_k \right]$$

donde:

$$p_k = \begin{cases} \left(\frac{1}{2}\right)^k & si \quad k = 1,2,\ldots,M-1 \\ \left(\frac{1}{2}\right)^{M-1} & si \quad k = M \end{cases}$$

Paso 5 Se pasa el test con un nivel de significancia del $\alpha\%$ si, y sólo si, $S \le \chi^2$, donde χ^2 es el mayor punto de $\alpha\%$ de la distribución con *M* - 1 grados de libertad.

Al igual que en el test anterior, se debe emplear el algoritmo de Massey-Berlekamp (que se describirá en el siguiente capítulo) para obtener el LCP de la secuencia. Si se testea la aleatoriedad de una secuencia de *n* bits $x_0, x_1,..., x_{n-1}$, este test sólo podrá realizarse si su LCP contiene al menos *F* saltos. Sin embargo, si el perfil tiene menos de *F* saltos, la secuencia ya habrá fallado al intentar pasar el test del número de saltos. Sin embargo, si se testea la secuencia obtenida por un generador es posible generar más bits de salida $x_n, x_{n+1}, x_{n+2},...$ hasta que se hayan obtenido los *F* saltos necesarios.

L) Test estadístico universal basado en la entropía por bit

En 1990, U.M. Mauer [MAU91] presentó un test estadístico para generadores de bits pseudoaleatorios basándose en los algoritmos universales de compresión de datos de Elias y de Willems, que pretende detectar cualquier desviación significativa de la estadística de una secuencia a través de medir la entropía por bit de salida del generador.

Este test ofrece principalmente dos ventajas sobre los tests estadísticos anteriores. En primer lugar, a diferencia de ellos, es capaz de detectar cualquiera de los diversos defectos estadísticos que pueda presentar el generador, incluídos todos los que detectaban los tests mencionados anteriomente. Más que medir determinados parámetros (como por ejemplo, la frecuencia relativa de unos o ceros), este test mide la significatividad de un defecto. Mauer mostró [MAU91] que el comportamiento no deseado de un generador de bits aleatorio podría modelarse adecuadamente como una fuente ergódica estacionaria con memoria relativamente pequeña.

El ejemplo más sencillo de un comportamiento estadístico defectuoso de este tipo puede ser una fuente binaria sin memoria, cuyos bits son estadísticamente e idénticamente (pero no necesariamente simétricamente) distribuídos, es decir, no son siquiera 1-distribuídos [MAUER]. BMS_p denotaría, entonces, esa fuente binaria sin memoria que emite unos con probabilidad p, y ceros con probabilidad $1 - p$. Otro ejemplo que podemos denotar como ST_p, sería una fuente binaria cuyos bits de salida estuvieran realmente distribuídos de forma simétrica ($p = 1/2$), pero sus probabilidades de transición estuviesen polarizadas: un dígito binario vendría seguido de su complemento con probabilidad p y por el mismo dígito con probabilidad $1 - p$.

Este sería un ejemplo de una fuente binaria estacionaria con un bit de memoria. En general, la probabilidad de distribución del bit i de salida de un generador puede depender de los M bits previos, donde M es la memoria de la fuente.

Evidentemente, ambos ejemplos no podrán generar secuencias k-distribuídas, ya que algunas combinaciones de k bits ($k > 1$ en el primer caso, y $k > 2$ en el segundo) serán preferentes frente a otras. Para realizar el test de entropía por bit (T_e) se debe dividir la secuencia que se quiere estudiar en bloques disjuntos de longitud L. La longitud total de la secuencia muestra x^N necesaria será de $N = (Q+K)L$ bits, donde K es el número de iteraciones que realizará el test, y Q el número de iteraciones iniciales que deben hacerse previamente a las K.

Supongamos que $b_n(x^N) = (x_{Ln},..., x_{Ln+L-1})$ con $0 \le n \le Q + K - 1$ denota el bloque n-ésimo de longitud L de la secuencia x^N. Para $Q \le n \le Q + K - 1$, se debe inspeccionar la secuencia en busca de la más reciente ocurrencia del bloque $b_n(x^N)$, es decir, el menor entero positivo i ($i \le n$), tal que $b_n(x^N) = b_{n-i}(x^N)$. Diremos que $A_n(x^N) = i$, si existe un bloque que lo cumpla, y si no fuera así se hará la asignación $A_n(x^N) = n$ (donde $A_n(x^N)$ se define en (9)).

La función de distribución para este test $f_{T_e}(x^N)$, y que transformará la secuencia binaria de longitud N en los números reales R, se define como la media del logaritmo (en base 2) de los K términos $A_Q(x^N)$, $A_{Q+1}(x^N)$,..., $A_{Q+K-1}(x^N)$, o, formalmente:

$$f_T(x^N) = \frac{1}{K} \sum_{n=Q}^{Q+K-1} log_2[A_n(x^N)]$$

donde para $Q \le n \le Q + K - 1$, $A_n(x^N)$ viene definida por:

$$A_n\left(x^N\right) = \begin{cases} n \quad si \quad existe \quad un \quad i \le n \quad tal \quad que \quad b_n\left(x^N\right) = b_{n-i}\left(x^N\right) \\ min\left\{i: i \ge 1, b_n\left(x^N\right) = b_{n-i}\left(x^N\right)\right\} \quad en \quad cualquier \quad otro \quad caso \end{cases} \tag{9}$$

Este test puede desarrollarse utilizando una tabla de tamaño 2^L que guarde, para cada combinación de L bits, el índice del tiempo de su ocurrencia más reciente. En la notación que se ha usado a lo largo de este libro se puede expresar como:

```
FOR i:= 0 TO 2^L-1 DO Tab[i]:= 0;
FOR n:= 0 TO Q -1 DO Tab[b_n(x^N)]:= n;
suma = 0,0;
FOR n:= Q TO Q + K - 1 DO
        BEGIN
        suma:= suma + log2(n- Tab[b_n(x^N)]);
        Tab[b_n(x^N)]:= n;
        END;
f_Te(x^N):= suma/K;
```

Se recomiendan los valores $8 \le L \le 16$, $Q \ge 5 \times 2^L$ y K tan grandes como sean posibles (por ejemplo $K = 10^4$ o $K = 10^5$) para realizar el test.

La elección de Q debería garantizar que, con probabilidad muy alta, cada subsecuencia de L bits ocurriese como mínimo una vez en los Q bloques de la secuencia, de manera que las tablas de $E[f_T(s^N)]$ y $VAR[\log_2[A_n(s^N)]]$ que se dan más abajo para valores de $Q \to \infty$ fuesen adecuadas para determinar los umbrales de detección t_1 y t_2. Si el valor de la entropía por bit sale fuera de estos umbrales, entonces se debe rechazar el generador como sospechoso de padecer algún defecto (aunque no sepamos cuál). La elección de estos umbrales plantea un compromiso entre la probabilidad de detección y la probabilidad de falsa alarma: si escogemos un margen demasiado pequeño entre t_1 y t_2 difícilmente ocurrirán falsas alarmas, pero tendremos también menor probabilidad de detectar algún generador defectuoso; si el margen entre t_1 y t_2 es muy amplio, detectaremos con mayor facilidad los problemas, pero probablemente muchas de esas detecciones serán simplemente falsas alarmas. Normalmente este compromiso se resuelve con posturas conservadoras respecto a la falsa alarma. Es aconsejable asegurarse de que la probabilidad de falsa alarma sea muy pequeña, del orden de 0,01 o de 0,001. O sea:

$$P(f_T(x^N) \le t_1) + P(f_T(x^N) \ge t_2) = \rho/2$$

donde ρ es la tasa de rechazo, y normalmente ocurre que $P(f_T(x^N) \le t_1) \approx \rho/2$. Para los valores umbrales se recomiendan los siguientes valores:

$$t_1 = E[f_T(x^N)] - y\partial \qquad t_2 = E[f_T(x^N)] + y\partial$$

donde $\partial = (\mathrm{VAR}[\log_2[A_n(x^N)]]/K)^{1/2} \approx (\mathrm{VAR}[f_{T_e}(x^N)])^{1/2}$, y donde y se escoge de forma que $N(-y) = \rho/2$. La función $N(x)$ es la integral de la función densidad de probabilidad normal definida como:

$$N(x) = (2\pi)^{-1/2} \int_{-\infty}^{x} e^{-f^2} dx$$

Una tabla de $N(x)$ se puede encontrar en casi cualquier libro de estadística o de teoría de probabilidad, pero para obtener tasas de rechazo de $\rho = 0{,}01$ y $\rho = 0{,}001$, deben tomarse $y = 2{,}58$ e $y = 3{,}30$ respectivamente.

Tabla 5.5 Esperanza y varianza de los valores que mide el test suponiendo $K \longrightarrow \infty$, $Q \longrightarrow \infty$.

L	$E[f_t(x^N)]$	$VAR[\log_2[A_n(x^N)]]$
1	0,73264948	0,690
2	1,53743829	1,338
3	2,40160681	1,901
4	3,31122472	2,358
5	4,25342659	2,705
6	5,21770525	2,954
7	6,19625065	3,125
8	7,18366555	3,238
9	8,17642476	3,311
10	9,17232431	3,356
11	10,1700323	3,384
12	11,1687649	3,401
13	12,1680703	3,410
14	13,1676926	3,416
15	14,1674884	3,419
16	15,1673788	3,421

Un generador debería pasar este test si, y sólo si, la secuencia de salida no puede comprimirse significativamente. En la tabla 5.5 se muestran la esperanza de $f_T(x^N)$ y una buena aproximación de su varianza, necesarias para realizar una implementación práctica de este test.

La entropía por bit H_x de la secuencia $\{x^N\}$ y la $E[f_T(x^N)]$ están relacionadas por la siguiente relación de convergencia [MAU91]:

$$\lim_{L \to \infty} \frac{E[f_{T_e}(x^N)]}{L} = H_x$$

5.2.4 Tests estadísticos teóricos

Aunque siempre es posible testear la aleatoriedad de las secuencias producidas por los generadores de secuencias pseudoaleatorias empleando los tests estadísticos empíricos descritos anteriormente, a veces puede ser interesante disponer, a priori, de algunos resultados que, obtenidos mediante procedimientos teóricos, nos den una información preliminar de cómo se comportarán las secuencias respecto a esos tests empíricos.

Los tests de tipo teórico nos darán además mayor información acerca del método que se ha utilizado para la generación de la secuencia que los tests de tipo empírico, ya que estos últimos consideran al generador como una caja negra y basan, por tanto, sus resultados en el método de la prueba y el error.

Otra diferencia fundamental de los tests teóricos respecto a los empíricos, es que para ser válidos deben considerar el periodo completo de la secuencia. Si bien los tests empíricos también se pueden realizar sobre todo el periodo de la secuencia, recordemos que ello no es necesario, aunque tanto mayor será la fiabilidad del test cuanto mayor sea la longitud de la muestra estudiada.

Puesto que la realización de los tests teóricos conlleva una importante complejidad matemática, que va mucho más allá del interés eminentemente práctico de esta obra, si el lector desea profundizar más en este tipo de tests puede recurrir a ciertas publicaciones que tratan con más en profundidad este tema.

Sin embargo, aquí se va a mostrar un importante test de aleatoriedad propuesto en 1965 por Coveyou y MacPearson, el test espectral, que contiene conceptos tanto de los tests estadísticos teóricos, ya que considera cantidades promediadas sobre todo el periodo de la secuencia, como de los empíricos, ya que requiere la utilización de un ordenador para proporcionar sus resultados.

A) El test espectral

Este test presenta una motivación matemática basada en la transformada de Fourier de una función definida sobre un conjunto finito. Dada una función $F(t_1, t_2,..., t_n)$ de valores complejos definida para todas las combinaciones de enteros t_k, donde $0 \leq t_k \leq m$ para $1 \leq j \leq n$, se define la transformada de Fourier de F mediante la siguiente expresión:

$$f(s_1, s_2, \ldots, s_n) = \sum_{0 \leq t_1, \ldots, t_n < m} exp(-2\pi i / m(s_1 t_1 + \ldots + s_n t_n)) F(t_1, t_2, \ldots, t_n)$$

Esta función f se define para todas las combinaciones de enteros s_k y será periódica. El nombre de transformada se justifica por el hecho de que la función original $F(t_1, t_2,..., t_n)$ puede reconstruírse a partir de su transformada mediante:

$$F(t_1, t_2, \ldots, t_n) = (1/m)^n \sum_{0 \leq s_1, s_2, \ldots, s_n < m} exp(-2\pi i / m(s_1 t_1 + \ldots + s_n t_n)) f(s_1, s_2, \ldots, s_n)$$

Esta fórmula puede desarrollarse en senos y cosenos y dar lugar a las series infinitas de Fourier.

El valor $(1/m)^n f(s_1,..., s_n)$ representa esencialmente la amplitud en un plano complejo *n*-dimensional de ondas con frecuencias $s_1/m,..., s_n/m$, si $F(t_1, t_2,..., t_n)$ se escribe como una superposición de dichas ondas.

Para aplicar estos conceptos al estudio de la generación de números aleatorios, supongamos que $X_0, X_1, X_2,...$ es una secuencia infinita de enteros con $0 \le X_k < m$, y sea *n* un número entero fijo (y normalmente bastante pequeño).

Se define $F(t_1, t_2,..., t_n)$ como:

$$F(t_1,t_2,\ldots,t_n) = \lim_{N\to\infty} (1/N) \sum_{0\le k<N} \partial_k t_1 \cdot \partial_{k+1} t_2 \ldots \partial_{k+n} t_n \tag{10}$$

esto es, $F(t_1, t_2,..., t_n)$ es la función densidad de ocurrencias de la subsecuencia de *n* términos (t_1, $t_2,..., t_n$) como *n* elementos consecutivos de la secuencia $X_0, X_1, X_2, ...$

Puesto que las secuencias que testeamos son periódicas de periodo *p*, podemos asumir que ese límite existe.

En una secuencia infinita verdaderamente aleatoria (que cumplirá *k*-distribución, y cada posible subsecuencia de *n* términos ($n < k$) aparecerá igualmente probable), $F(t_1, t_2,..., t_n)$ debería ser $(1/m)^n$ para todo $t_1, t_2,..., t_n$ La transformada de (10) se podrá expresar como:

$$f(s_1,\ldots,s_n) = \lim_{N\to\infty} (1/N) \sum_{0\le k<N} exp(-2\pi i / m(s_1 X_k + \ldots + s_n X_{k+n-1})) \tag{11}$$

Una secuencia verdaderamente aleatoria debería ser la transformación de la función constante $(1/m)^n$. Por tanto, deberíamos encontrar:

$$f(s_1,\ldots,s_n) = \begin{cases} 1 & si \cdot s_1 = \ldots = s_n = 0 \cdot mod \cdot m \\ 0 & en \cdot otro \cdot caso \end{cases}$$

Un ejemplo práctico de implementación vendría determinado por secuencias de tamaño igual a un periodo (*p*). Si se trata de secuencias verdaderamente *k*-distribuídas tendrán por transformada:

$$f(s_1,\ldots,s_n) = \begin{cases} 1 & si \cdot s_1 = \ldots = s_n = 0 \cdot mod \cdot m \\ 1/p & en \cdot otro \cdot caso \end{cases} \tag{12}$$

Así pues, este test consistirá en calcular la transformada (11) para la secuencia de test y comprobar cuánto se acerca al resultado esperado (12). Una buena forma de medir cuán cerca se está de este resultado es medir la relación lóbulo principal-lóbulo secundario (LPLS) en decibelios, utilizando para ello la expresión:

$$LPLS = 10 \cdot log\left(f_{max} / f_{lob\_sec} \right)$$

5.3 Problemas

PROBLEMA 5.1 Si se trucan dos dados, de forma que en uno de ellos el valor 1 pase a tener exactamente el doble de probabilidad que cualquier otro valor, y el otro dado se truca igualmente, pero en este caso el número con doble probabilidad es el 6, calcular la probabilidad p_s de que la suma de los dos dados sea s, para $2 \le s \le 12$.

PROBLEMA 5.2 Unos dados trucados, como en el ejercicio anterior, se tiran 144 veces y el número de veces Y_s que aparece cada uno de los posibles valores de la suma s son los siguientes:

s	2	3	4	5	6	7	8	9	10	11	12
Y_s	2	6	10	16	18	32	20	13	16	9	2

Sobre estos resultados:

a) Aplicar un test chi-cuadrado sobre estos valores, usando las probabilidades del ejercicio del apartado 5.2.2 A, suponiendo que el hecho de que los dados estén trucados no es conocido.
b) ¿Sirve el test chi-cuadrado para detectar dados trucados?
c) Si no es así, explicar por qué.

PROBLEMA 5.3 Mostrar por qué las estadísticas K^+_n y K^-_n del test de Kolmogorov-Smirnov no pueden ser nunca negativas. ¿Cuál es el máximo valor que puede valer K^+_n?

PROBLEMA 5.4 Determinar el algoritmo para extender los tests sucesivos a la medida de la equidistribución de quíntuplas.

PROBLEMA 5.5 ¿Cuántos bits serán necesarios para examinar el test de brechas antes de que se encuentren n brechas, en media, asumiendo que la secuencia es aleatoria?

PROBLEMA 5.6 Aplicar el test colector de cupones con $c = 3$ y $n = 7$ a la siguiente secuencia: 11012210221202020012122010102011 21. ¿Qué longitudes tienen las 7 subsecuencias?

PROBLEMA 5.7 ¿Cuántos bits se deben examinar en media en el test colector de cupones antes de que se encuentren n conjuntos, asumiendo que la secuencia es aleatoria?

PROBLEMA 5.8 Generalizar el test colector de cupones de forma que la búsqueda finalice tan pronto como se hayan encontrado w valores distintos, donde w es un entero fijo positivo menor o igual que c. ¿Qué probabilidades se deben emplear?

PROBLEMA 5.9 Sean $x_0, x_1, ..., x_{n-1}$, n números distintos. Escribir un algoritmo que determine la longitud de todas las subsecuencias descendentes de la secuencia, usando las siguientes indicaciones: cuando el algoritmo termine su ejecución, CONT[r] debe ser el número de subsecuencias de longitud r,

para $1 \leq r \leq 5$, y CONT[6] debe ser el número de subsecuencias de longitud mayor o igual que 6.

PROBLEMA 5.10 Para el coeficiente de correlación serie mostrar que:

(a) Si $n = 2$, el coeficiente de correlación sucesiva es siempre -1 (a menos que el denominador sea cero).
(b) Si $n = 3$, el coeficiente de correlación sucesiva es siempre -1/2.
(c) El denominador de la expresión (8) del apartado 5.2.3. es cero si, y sólo si, $x_0 = x_1 = ... = x_{n-1}$.

PROBLEMA 5.11 ¿Cuál es la media y la desviación típica del coeficiente de correlación sucesiva cuando $n = 4$ y las x´s son independientes y están uniformemente distribuídas entre 0 y 1?.

PROBLEMA 5.12 Si X es una variable aleatoria con una distribución chi-cuadrado determinar:

a) $P(X \geq 26{,}1)$ si $\nu = 14$.
b) $P(X \leq 2{,}71)$ si $\nu = 1$.
c) $P(X < 21{,}6)$ si $\nu = 17$.

PROBLEMA 5.13 Si se han obtenido las siguientes observaciones de una variable aleatoria X: -2, 0, 6, 4, -1, -5, 12, 13, 3 y 11, calcular la media de las muestras y su desviación estándar.

PROBLEMA 5.14 Si suponemos que $\sigma = 6$ y $\alpha = 0{,}05$, comprobar las siguientes hipótesis para los datos obtenidos en el ejercicio anterior.

a) $H_0 : \mu = 4$, $A : \mu \neq 4$.
b) $H_0 : \mu = 6$, $A : \mu \neq 6$.
c) $H_0 : \mu < 5$, $A : \mu > 5$.

PROBLEMA 5.15 Se toman las siguientes muestras de dos variables aleatorias:

X : {2, 4, 6, 7, 9, 14, 16, 3} e Y: {-1, 4, 1, 6, 12, 12, 1, 0}

Si se asume que ambas presentan iguales varianzas, comprobar la hipótesis de que también tengan iguales medias con $\alpha = 0{,}05$.

PROBLEMA 5.16 Comprobar la suposición que se ha hecho en el ejercicio anterior de que ambas muestras presentan igual varianza.

PROBLEMA 5.17 Suponemos que los mensajes que llegan a un receptor siguen una distribución de Poisson, y que el intervalo entre sus llegadas sigue una distribución exponencial. Testear esta hipótesis mediante la utilización de un test chi-cuadrado, suponiendo que se dispone de la siguiente tabla de frecuencia de los tiempos entre llegadas de mensajes:

Tiempo	0-1	1-2	2-3	3-4	4-5	5-6	6-7
Frecuencia	2	6	12	16	21	11	6

PROBLEMA 5.18 Testear la hipótesis del ejercicio anterior mediante el test de Kolmogorov-Smirnov.

PROBLEMA 5.19 Se ha obtenido la siguiente tabla de una variable aleatoria *X*.

Intervalo	0-5	5-10	10-15	15-20	20-25	25-30	30-35	35-40
Frecuencia	0	2	2	6	10	20	28	32

Intervalo	40-45	45-50	50-55	55-60	60-65	65-70	70-75
Frecuencia	32	28	18	12	6	4	0

Comprobar la hipótesis de que X proviene de una distribución normal usando:

a) El test chi-cuadrado.
b) El test de Kolmogorov-Smirnov.

5.4 Bibliografía

[BER68] BERLEKAMP, E. R. *Coding Theory*. McGraw-Hill, 1968.

[CAN88] CANAVOS. *Probabilidad y estadística. Aplicaciones y Métodos*. McGraw-Hill. Mexico. 1988.

[CAR89] CARTER, G. *Statistical Tests for Randomness*. Proceedings of the Workshop on Stream Ciphers. E.I.S.S. 1989.

[CHA87] CHAITIN, G. J. *Information, Randomness and Incompleteness.* World Scientific Publishing, Singapore, 1987.

[CRA70] CRAMER. *Métodos matemáticos de estadística*. Aguilar S.A. Madrid 1970.

[FRE75] FREDRICSSON, S. *Pseudo Randomness Properties of binary Shift Register Sequences.* IEEE Trans. Inform theory, Vol. IT-21, pp. 115-120, 1975.

[GOL65] GOLD, R.; KOPITZKE, E. *Study of Correlation Properties of Binary Sequences.* Interim. Tech. Rep. 1, Vols 1-4, Magnevox REs. Lab. Torrance, CA, 1965.

[GOL67] GOLOMB, S. W. *Shift Register Sequences*. Holden-Day. 1967.

[GOL68] GOLOMB, S. W. *Theory of Transformation Groups of Polynomials Over GF(2) with Applications to Linear Feedback Shift Registers Sequences*. Inform Sci., V. 1, pp. 87, 1968.

[GOL90] GOLIC. *A Number of Output Sequences of a Binary Sequence Generator*. Proc. Crypto 89. Springer-Verlag Lecture Notes in Computer Science, No 435, New York, 1990.

[GRAYB] GRAYBEAL, W. J.; POOCH, U. W. *Simulation: Principles and Methods*. Winthop Publishers, Inc, Cambridge, Massachusetts.

[HEL76] HELLESETH, T. *Some Results About the Cross-Correlation function Between Two Maximal Linear Sequences*. Discrete Math. Vol. 16, pp. 209-232, 1976.

[KNU67] KNUTH, D. E. *The Art of Computer Programming. Vol 1: Seminumérical Algorithms*. Adison-Wesley 1967.

[KNUTH] KNUTH, D. E. *The Art of Computer Programming. Vol 2: Seminumérical Algorithms*. Adison-Wesley 1967.

[LID83] LIDL, R.; NEDERREITER, H. *Finite Fields*. Addison-Wesley, Reading Mass, 1983.

[MAR66] MARTIN-LÖF,P. *The Definition of Random Sequences*. Inform. Contr., Vol. 9, pp. 602-619, 1966.

[MAS76] MASSEY, J. L. *Shift Register Sysnthesis and BCH Decoding*. IEEE Trans on Information Theory, Vol. IT-15, No 1, Jan. 1976.

[MAU91] MAUER, U. *A Universal statistical Test for Random Bit Generators*. Advances in Cryptology- CRYPTO´90. Springer-Verlag Lecture Notes in Computer Science, No 537. Berlin, 1991.

[MAUER] MAUER, U.; MASSEY, J. L. *Perfect Local Randomness in Pseudo-random Sequences*. Proceedings Crypto 89, Santa Barbara.

[MOO40] MOOD, A. M. *The Distribution Theory of runs*. Ann. Math. Statist., 11, 1940, pp 367-392.

[MOO68] MOON, J. W.; MOSER, L. *On the Correlation Function of Random Binary Sequences.* SIAM J. Appl. Math, Vol. 16, Nº2, pp. 340-342, 1968.

[NIE88] NIEDERRETER, H. *The Probabilistic Theory of Linear Complexity*. Advances in Cryptology: Proceedings of Eurocrypt 88, Springer-Verlag, Berlin, 1988, pp. 191-209.

[PUR77] PURSLEY, M. B.; SARWATE, D. V. *Evaluation of Correlation Parameters for Periodic Sequences*. IEEE Trans. on inform theory, Vol. IT-23, pp. 508-513, 1977.

[RUE86] RUEPPEL, R. A. *Linear Complexity and Random Sequences*. Proceeding of Eurocrypt´86, Lecture Notes in computer Science 219, Springer-Verlag, 1986.

[SAR80] SARWATE, D.; PURSLEY, M. B. *Crosscorrelation Properties of Pseudo Random and Related Sequences.* Proc. of the IEEE, Vol. 68. Nº 5, May 1980.

[SCH89] SCHNORR, C. P. *Efficient, perfect random number generation*. Proccedings of the Workshop on stream Ciphers. Report 89/1, 1989.

[SID69] SIDELNIKOV, V. M. *Some k-valued Pseudo Random Sequences and Nearly Equidistant Codes*. Probl. Inform Transmission, Vol. 5, pp. 21-16, 1969.

[WEL74] WELCH, L. R. *Lower Bounds on the Maximum Cross-Correlation of Signals*. IEEE Trans. Inform theory, Vol. IT-20, pp. 397-399, 1974.

6 Complejidad de secuencias

6.0 Introducción

En el capítulo anterior se vieron los métodos para comprobar las propiedades estadísticas de las secuencias pseudoaleatorias. Sin embargo, el hecho que las secuencias muestren un buen resultado respecto a los tests que se presentaron en dicho capítulo, no garantiza que la secuencia sea impredictible. Tal como se vio en el capítulo 1, la impredictibilidad es una característica inherente al concepto de aleatoriad, y deberemos ser capaces de evaluarla.

Una medida importante de la impredictibilidad de una secuencia es su complejidad, de forma que para que la podamos considerar como impredictible, deberá mostrar una elevada complejidad y, además, cualquier subsecuencia de la secuencia considerada deberá mostrar también una complejidad elevada. Esta última característica determina lo que llamaremos el perfil de complejidad.

También se verá que, si bien la complejidad lineal y el perfil de complejidad lineal son las medidas de impredictibilidad más utilizadas, debido a que existe un algoritmo eficiente para calcularlas, existen otras medidas de complejidad de gran utilidad como la complejidad de máximo orden o la complejidad de Ziv-Lempel.

Estas últimas ofrecerán información adicional, que nos permitirá obtener conclusiones más fiables sobre la impredictibilidad de la secuencia considerada que si se utiliza únicamente la complejidad lineal.

A pesar de la importancia del concepto de impredictibilidad en las secuencias aleatorias, habrá que tener en cuenta que para algunas aplicaciones de telecomunicaciones que emplean este tipo de secuencias, éste no será un requisito imprescindible, y bastará que se usen secuencias con buenas propiedades respecto a los tests estadísticos vistos en el capítulo anterior.

Algunos ejemplos de este tipo de aplicaciones son las modulaciones *spread spectrum*, las técnicas de acceso múltiple por división de código (CDMA) y la aleatorización de las comunicaciones en enlaces vía satélite o cable. Sin embargo, habrá otras aplicaciones en las que la complejidad de las secuencias será un requisito absolutamente imprescindible, y el sistema quedará completamente invalidado si las secuencias no lo cumplen. Una aplicación de estas características es el cifrado en flujo.

De hecho, la enorme importancia que tiene el requisito de impredictibilidad en este campo es el que ha impulsado en gran medida el desarrollo de la teoría de la complejidad. Todas estas aplicaciones se verán detalladamente en el capítulo 7.

6.1 La complejidad de secuencias pseudoaleatorias

Se puede afirmar de una forma poco precisa, que la complejidad de una secuencia finita es una medida de cuánto se parece dicha secuencia a una secuencia verdaderamente aleatoria. El concepto de complejidad, tal como ya se ha ido viendo a lo largo de los capítulos anteriores, se podrá utilizar como medida del grado de impredictibilidad que presenten las secuencias, y estará además íntimamente relacionado con la compresión de secuencias.

Sin embargo, no existe un acuerdo claro en la literatura al definir la complejidad de las secuencias, y ello ha llevado a que aparezcan diversas definiciones de esta complejidad. Los primeros trabajos en este área [KOL65][MAR66] consideraban la complejidad de una secuencia en función del algoritmo que la generaba.

A.N. Kolmogorov [KOL65] propuso utilizar la longitud del programa más corto capaz de generar la secuencia como medida de su complejidad. A este programa se le denomina Máquina de Turing (TM), y a la longitud del programa más corto que sea capaz de generar una secuencia *S* determinada se le denomina complejidad de Turing-Kolmogorov-Chaitin de dicha secuencia *S* [LEU85].

Posteriormente, han ido apareciendo trabajos que, manteniendo en mayor o menor grado el concepto inical de complejidad de Kolmogorov y Chaitin, han ido refinando sucesivamente este concepto [WAH71][ASS74][CHA74]. Algunas de las definiciones más avanzadas en esta línea es la que hacen A.Lempel y J.Ziv en 1976 [LEM76], cuando proponen una aproximación que relaciona el concepto de complejidad de una secuencia con la aparición gradual de nuevos patrones a lo largo de dicha secuencia. Con esta aproximación se abren dos nuevas puertas. Por un lado, un nuevo método de construir secuencias que parezcan aleatorias, y por otra parte, una nueva forma de conseguir la compresión de secuencias de una forma eficiente. Lempel y Ziv proponen evaluar la complejidad de una secuencia finita desde el punto de vista de una simple máquina de aprendizaje autodelimitada que, a medida que va examinando una determinada secuencia de n dígitos $S = s_1, s_2, ..., s_n$ de izquierda a derecha, añade una nueva palabra a su memoria cada vez que descubre una subsecuencia de dígitos consecutivos que no había encontrado previamente. El tamaño del vocabulario obtenido, así como la tasa a la que se van encontrando nuevas palabras a medida que se estudia la secuencia *S*, son los elementos básicos que definen esta complejidad que llamaremos de Ziv-Lempel y que estudiaremos más adelante. En este caso, el concepto de complejidad está relacionado con la velocidad de crecimiento del vocabulario.

Sin embargo, a pesar de estos intentos por definir la complejidad de una secuencia, recientemente han aparecido nuevas propuestas, de entre las cuales la más comúnmente aceptada es la de la complejidad lineal [RUE86]. Esta complejidad se define como la longitud del registro de desplazamiento con realimentación lineal (LFSR) más corto que es capaz de generar la secuencia dada, es decir, determina qué porción de subsecuencia es necesaria para ser capaz de reproducir toda la secuencia. Como existe un algoritmo eficiente para calcular el LFSR más corto capaz de generar la secuencia, esta medida de la complejidad ha adquirido una gran importancia como medida de impredictibilidad de secuencias. Este algoritmo es el que Massey y Berlekamp propusieron en 1969 [MAS76], y su propósito original no estaba directamente relacionado con la medida de la complejidad de secuencias sino con la codificación. Fue posteriormente cuando se encontró la aplicación de este algoritmo para la síntesis de registros de desplazamiento con realimentación lineal (LFSR). Este

algoritmo, además de computar la complejidad lineal de una secuencia, también es capaz de determinar el polinomio de realimentación único y el estado inicial de un registro de desplazamiento que será capaz de generarla completamente. En el artículo original, el algoritmo no se restringe a los números binarios sino que puede utilizarse para números reales.

Existen otras medidas de complejidad que van más allá de la definición lineal, como la complejidad de máximo orden que propuesta por Jansen y Boekee [JAN89], y que define la complejidad de la secuencia como el registro de desplazamiento con realimentación, pero esta vez no lineal, que es capaz de generar la secuencia.

En los próximos apartados vamos a ver más detalladamente todos estos conceptos de complejidad.

6.1.1 La complejidad de Turing-Kolmogorov-Chaitin (TKC)

Un autómata determinista *M* con entrada/salida binaria se suele representar generalmente por [LEU85]:

$$M = (S, \Sigma, \Omega, \delta, \alpha, \lambda)$$

donde *S* representa el conjunto de estados con un estado inicial especial α. Los alfabetos en entrada (I) y de salida (O) coinciden con $\Sigma = \{0,1\}$. $\delta: S \times I \rightarrow S$ representa la función de siguiente estado de la máquina, y $\lambda: \Omega \rightarrow O$ denota la función de salida, que producirá una salida siempre que la máquina alcance un estado que pertenezca al conjunto de estados finales $\Omega \subset S$.

Se dice que la secuencia $\underline{x} \in 2^n$ estará generada por la máquina *M* si existe una secuencia $(w_1, w_2, \ldots, w_n) \in \Omega^n$ en la que se puede llegar a w_{i+1} a partir de w_i para *i* = 1, 2,..., *n* - 1 y donde

$$\lambda(w_i) = x_j \qquad \textit{para} \qquad j \in [1, 2, \ldots, n]$$

Entonces se dice que la máquina *M* es un generador de secuencias para todo $n \in N$.

Se puede considerar una Máquina de Turing como una máquina de estados finitos con un dispositivo adicional de almacenamiento ilimitado, que actúa como espacio de trabajo donde se podrán escribir o leer los símbolos *Z* = {0,1,#}.

El cabezal que realizará estas escrituras y lecturas estará controlado por un control de estado finito a través de los comandos +1 (izquierda), -1 (derecha) y 0 (parar). La descripción formal de esta máquina de estados finitos vendrá dada por:

$$T = (S, I, Z; \delta, \alpha, \Omega)$$

La función del próximo estado de la máquina δ se puede extender como la función que determina el siguiente movimiento:

$$\delta: S \times I \times Z \rightarrow S \times \left(Z \times \{Z_3\}\right)$$

de forma que a partir de un estado determinado de la máquina de estados $\sigma \in S$, se puede computar un símbolo de entrada $i \in I$ y un símbolo de salida (leído) $z \in Z$

$$\delta(\sigma, i, z) = (\sigma', z', m)$$

Por ejemplo, si el estado de la máquina cambia de σ a σ', se escribirá un nuevo símbolo en el dispositivo de almacenamiento y se desplazará la cabeza de lectura/escritura en $m \in Z_3$ (a la derecha, a la izquierda, o nada). Al comienzo, el dispositivo de almacenamiento debe estar vacío, por ejemplo lleno con espacios blancos (#).

El concepto de una máquina de Turing se puede, pues, usar para construir la llamada Máquina Universal de Turing (UTM), que en esencia es una Máquina de Turing de propósito general capaz de simular cualquier otra Máquina de Turing como la descrita anteriormente.

Una Máquina Universal de Turing será una máquina de Turing *U*, cuyo control de estados finitos se diseña de forma que genere secuencias binarias de salida $\underline{x} \in \{0,1\}^*$ de la forma $\underline{y} = (\underline{x}, \underline{p}(T))$ donde $\underline{p}(T)$ sea una entrada igual al programa binario, que haga que *U* se comporte como la Máquina de Turing *T* que genere la secuencia $\underline{x}$. A continuación se define la complejidad TKC utilizando para ello algunos teoremas:

TEOREMA 1: Cualquier Máquina de Turing *T* se puede simular de esta forma mediante una Máquina Universal de Turing *U*.

TEOREMA 2: Sea $\underline{x} \in 2^n$ una secuencia binaria. Sea *T* una máquina de Turing determinada que genere la secuencia $\underline{x}$, y sea $\underline{p}(T)$ el programa que hace que esta UTM simule a *T*. Entonces el número $\mu(T) = \left|\underline{p}(T)\right|$, se llamará el tamaño de *T*, donde $\left|\underline{p}\right|$ será, por ejemplo, la longitud del programa p. Además, el número:

$$\chi(\underline{x}) = \mu_\Gamma(\underline{x}) = min\{\mu(T) \mid T \quad genera \quad \underline{x}, T \in \Gamma\}$$

se le llama la complejidad de Turing-Kolmogorov-Chaitin de la secuencia. Γ es la clase de Máquinas de Turing.

TEOREMA 3: La función χ generalmente no es computable (mediante una máquina de Turing).

TEOREMA 4: En el espacio 2^n de todas las secuencias binarias con probabilidad equidistribuída, para todo ε tal que se cumple que $0 < \varepsilon < 1$:

$$Prob\{\underline{s} \in 2^n \mid \chi(\underline{s}) > (1-\varepsilon)n\} > 1 - 2^{-\varepsilon n + 1}$$

La conclusión de este teorema es que prácticamente todas las secuencias de longitud moderadamente larga presentan una complejidad de Turing-Kolmogorov-Chaitin muy cercana a la longitud de la secuencia o, dicho de otra forma, una secuencia verdaderamente aleatoria de longitud *n*

no puede tener una descripción más corta que la propia secuencia. Una frase que podría resumir de forma breve todos estos teoremas sería que si somos capaces de describir una secuencia, ésta no podrá ser aleatoria [BET89].

La introducción del concepto de Máquina de Turing aparece como consecuencia del hecho de que no hay forma posible de generar un gran subconjunto de secuencias binarias de longitud *n* que parezcan verdaderamente aleatorias mediante una máquina de estados finitos comparativamente pequeña. En otras palabras, los resultados anteriores muestran que el uso de los llamados generadores no lineales no aporta mucho en términos de la complejidad de las secuencias, si se evalúa la complejidad de dichos generadores de forma correcta.

La imposibilidad de computar la complejidad de Turing-Kolmogorov-Chaitin, ha llevado a buscar otros conceptos de complejidad más desarrollados y cuyo cómputo sea sencillo. Un concepto de complejidad que ha tenido gran difusión ha sido el de la complejidad lineal de las secuencias que vamos a ver a continuación.

6.1.2 La complejidad lineal (Λ)

Tal como se ha expuesto anteriormente, uno de los requisitos que deben cumplir las secuencias aleatorias es que deben mostrar una alta impredictibilidad. Para que la secuencia sea impredictible, su complejidad lineal debe ser alta. Esta es una condición necesaria, pero no suficiente, para garantizar la impredictibilidad de la secuencia. Por ejemplo, la secuencia de longitud 31 {00000000000000000000000000000001} es de máxima complejidad lineal, ya que según se puede observar, se necesita un LFSR de su longitud para poder generarla, pero no es utilizable como secuencia aleatoria puesto que es totalmente predecible, ya que no cumple el requisito de distribución estadística uniforme de ceros y unos. La impredictibilidad requiere que, independientemente del dígito precedente, el siguiente bit de la secuencia aparezca con una distribución estadística uniforme.

Una razón más para usar como parámetro de medida la complejidad lineal es que el periodo de la secuencia de salida del generador sea, al menos, tan grande como la complejidad lineal, por lo que una gran complejidad lineal implicará siempre un gran periodo.

Las secuencias pseudoaleatorias muestran un aumento de la complejidad lineal al aumentar el número de dígitos de la secuencia. Esto no ocurre a la inversa, pues una secuencia periódica no implica necesariamente gran complejidad lineal.

Las secuencias de longitud máxima o pseudoaleatorias obtenibles de los registros de desplazamiento con realimentación lineal (LFSR) tienen excelentes propiedades de distribución estadística, pero tienen complejidad lineal mínima respecto a la longitud del periodo, y se determinan completamente conociendo sólo $2L$ bits de la secuencia (L es el orden, o también el número de celdas del LFSR). Se puede generar una secuencia periódica por cualquier método (lineal o no), pero siempre será posible obtenerla a partir de un LFSR de longitud mínima, a cuyo orden llamaremos complejidad (o equivalente) lineal Λ.

Cuanto mayor sea el periodo o la complejidad lineal, mayor será la impredictibilidad de la secuencia. Con un LFSR se estará siempre limitado a una complejidad menor o igual a su orden, es decir, $\Lambda \leq L$. Por ello usaremos funciones no lineales que permitan alcanzar complejidades lineales $\Lambda > L$. Siempre se podrá obtener una secuencia a partir de un LFSR de orden mayor que la

complejidad lineal pero no será práctico, pues se estará usando más circuitería que la estrictamente necesaria sin obtener a cambio ninguna ventaja. Algunas conclusiones sobre complejidad lineal que parecen ampliamente aceptadas son:

1) Por convenio, se dice que la complejidad de la secuencia todo ceros es cero.

2) La secuencia todo ceros y un uno (como en el ejemplo anterior) tiene máxima complejidad lineal (en el ejemplo complejidad $\Lambda = 31$), puesto que sólo un LFSR de $L = 31$ puede generar esta secuencia.

3) La máxima complejidad que puede alcanzar una secuencia de longitud n es n, y la de una secuencia periódica de periodo P es P.

4) La adición al LFSR de un filtro de estado no lineal aumenta considerablemente la complejidad. Pero siempre hay que comprobar que se mantienenen las buenas propiedades estadísticas.

El punto 2 merece un comentario adicional. El hecho de que la secuencia todo ceros y un uno presente una elevada complejidad lineal, no implica en absoluto que ésta sea de utilidad desde el punto de vista de la aleatoriedad, debido a la pésima distribución de unos y ceros que presenta. Esto demuestra que altas complejidades son necesarias pero no suficientes, y que habrá que tener un cuidado extremo a la hora de diseñar funciones no lineales con el propósito de aumentar la complejidad de las secuencias para que no estropeen las propiedades de distribución de ceros y unos. Del mayor o menor éxito que se consiga en este requisito dependerá, en gran medida, la aplicabilidad del generador que se obtenga. En el capítulo 4 se vieron numerosas técnicas que, a partir de m-secuencias que presentan excelentes distribuciones estadísticas, aunque baja complejidad, tenían como objetivo aumentar esta última sin estropear la primera, introduciendo para ello algún tipo de no aleatoriedad.

Volviendo al hecho de que una elevada complejidad lineal no implica necesariamente la utilidad de la secuencia, se debe llevar más allá el concepto de complejidad para detectar casos como el anterior. Esto nos lleva a la necesidad de analizar el comportamiento del crecimiento de dicha complejidad [DAI86].

6.1.3 El perfil de complejidad lineal (LCP)

Para poder usar la complejidad lineal como parámetro de medida de la impredictibilidad de las secuencias, debemos afinar más en sus valores y en su comportamiento. No sólo será importante el valor final que alcanza la complejidad, sino también la forma de crecimiento que tiene ésta al ir procesando sus bits. Para el ejemplo del apartado anterior (secuencia con 30 ceros y un 1), la complejidad da un único salto de valor 31, lo cual no se corresponde en absoluto con una distribución uniforme cuya complejidad aumentará de forma gradual ajustándose a unos límites a lo largo de toda la secuencia. El perfil de crecimiento de la complejidad (o LCP según las siglas en inglés), estudiada por

Ruepple [RUE86], es útil para detectar secuencias cuya complejidad es buena, pero que no son recomendables para ser utilizadas en aplicaciones como la criptografía.

Sea $\Lambda(S_n)$ la complejidad lineal resultante del cómputo de la complejidad lineal de los primeros *n* dígitos de una secuencia aleatoria *S*. Si se representa gráficamente el crecimiento de la complejidad lineal en función del número de bits calculados hasta ese momento, para ver el comportamiento de la misma en lo que se refiere a saltos bruscos de su valor tendremos:

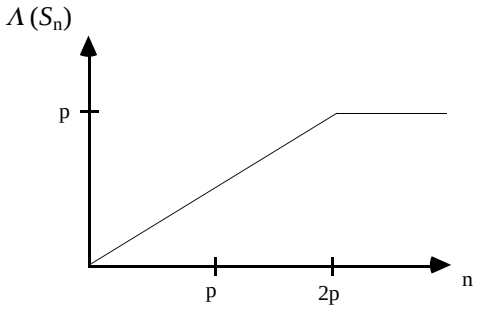


Fig. 6.1 Perfil de complejidad lineal de una secuencia pseudoaleatoria

En esta gráfica (figura 6.1) se puede observar como, para secuencias verdaderamente aleatorias, la complejidad crecerá siguiendo, aunque irregularmente, la línea de *N*/2, donde *n* es el número de bits de la secuencia que se han procesado.

Se puede, por tanto, caracterizar el crecimiento de la complejidad lineal como una clase especial de paso aleatorio (en el capítulo 5 se vieron dos tests estadísticos basados precisamente en este hecho).

Puesto que las secuencias que obtendremos en la práctica serán pseudoaleatorias, debido a su naturaleza periódica, deberemos intentar que al igual que para las secuencias verdaderamente aleatorias, su complejidad crezca ciñiéndose lo máximo posible a la línea de *n*/2, de forma que si el periodo de la secuencia es *p*, la complejidad alcance este valor después de procesar dos periodos completos de la secuencia. Por supuesto, el que la complejidad llegue a crecer hasta el valor del periodo dependerá, como ya sabemos, de lo bien diseñado que esté el generador y, en cualquier caso, nunca podrá ir más allá del periodo de la secuencia.

Para una aplicación como el cifrado en flujo, que requiere que las secuencias empleadas presenten el mismo comportamiento respecto al LCP que las secuencias verdaderamente aleatorias, se deberá rechazar cualquier secuencia que se desvíe significativamente del perfil de crecimiento irregular siguiendo la línea de *n*/2.

Veamos cómo podemos obtener el número de secuencias de longitud *n* con complejidad lineal igual a su periodo (y, por tanto, la máxima obtenible). Dicho número lo vamos a denotar por $N_n(L)$. Hay que iniciar este estudio observando el crecimiento de la complejidad lineal al considerar nuevos dígitos de la secuencia. Para ello, llamaremos próxima discrepancia a la diferencia entre la *n*-ésima variable aleatoria binaria y el *n*-ésimo dígito de la secuencia generada por un LFSR. La discrepancia no

cambiará de valor cuando la $\Lambda \geq L$, puesto que con una realimentación lineal lo mejor que se puede esperar es alcanzar un valor de la complejidad lineal igual al orden del LFSR. Si se representan estas condiciones de forma unificada, si n es el número de bits computados, tendremos que:

$$\text{DISCREPANCIA} = 0 \quad \Rightarrow \quad \Lambda(s_n) = \Lambda(s_{n-1})$$

$$\text{DISCREPANCIA} = 1 \quad \Rightarrow \quad \begin{cases} \Lambda(s_n) = \Lambda(s_{n-1}) & \text{si } \Lambda(s_{n-1}) \geq n/2 \\ \Lambda(s_n) = n - \Lambda(s_{n-1}) & \text{si } \Lambda(s_{n-1}) < n/2 \end{cases}$$

De estas ecuaciones se puede obtener directamente el resultado para $N_n(L)$, que es el número de secuencias con un determinado valor de la complejidad lineal. Si se representa su valor de forma recursiva se obtienen las siguientes expresiones:

$$N_n(L) = \begin{cases} 2N_n(L) + N_{n-1}(n-L) & n \geq L > n/2 \\ 2N_{n-1}(L) & L = n/2 \\ N_{n-1}(L) & n/2 > L \geq 0 \end{cases}$$

si las condiciones iniciales de la recursión son $N_1(0) = 1$ y $N_1(1) = 1$.

Como punto de referencia se debe tener en cuenta que el número de secuencias diferentes existentes para una longitud n es de 2^n.

También se puede expresar el número de secuencias de longitud n con complejidad lineal L en una forma no recurrente:

$$N_n(L) = \begin{cases} 2^{\min(2n-2L,\,2L-1)} & n \geq L, L > 0 \\ 1 & n \geq 0 \end{cases}$$

Este valor de $N_n(L)$ es el número exacto de secuencias de longitud n con complejidad lineal L, pero nos puede interesar también conocer el número de secuencias N_L cuya complejidad lineal sea igual o menor que L. Para ello se puede usar la expresión:

$$N_L = 1 + \sum_{j=1}^{L} 2^{2j-1} = \frac{2}{3}2^{2L} + \frac{1}{3}$$

A partir de ella se puede observar que 2/3 de todas las posibles secuencias de longitud L pueden ser generadas por LFSRs de L o menos celdas.

Las secuencias de complejidad lineal L o menor se caracterizan por el hecho de que el LFSR asociado que la produce es único.

La mayoría de las secuencias aleatorias de longitud L tendrán una complejidad lineal próxima a $n/2$, aunque valores lejanos o próximos a n también serán posibles.

Se puede conocer cuál será el valor de la complejidad lineal esperada de una secuencia de n términos realizando un planteamiento estadístico a priori. De esta forma se obtiene la expresión:

$$E(A) = \frac{n}{2} + \frac{4+R}{18} - 2^{-n}\left(\frac{n}{3} + \frac{2}{9}\right) \cong \frac{n}{2} + \frac{4+R}{18}$$

donde R es el resto de la división de n entre 2. Esta esperanza tiende a $n/2$ cuando n es muy grande.

Dentro de la caracterización estadística que se está calculando sobre la complejidad lineal, además de la media o esperanza necesitamos conocer la varianza o dispersión respecto al valor medio. Este valor viene determinado por la siguiente expresión:

$$\mathrm{VAR}(\Lambda) = \mathrm{E}((\Lambda - \mathrm{E}(\Lambda))^2) = \mathrm{E}(\Lambda^2) - \mathrm{E}(\Lambda)^2 =$$

$$= \frac{86}{81} - 2^{-n}\left(\frac{14-R}{27}n + \frac{82-2R}{81}\right) - 2^{-2n}\left(\frac{1}{9}n^2 + \frac{4}{27}n + \frac{4}{81}\right)$$

Cuando $n \longrightarrow \infty$, esta varianza valdrá $\mathrm{VAR}(\Lambda) = 86/81$.

Como se ha dicho antes, una parte del éxito de la complejidad lineal como medida de la impredictibilidad de la secuencia reside en la existencia de un algoritmo eficiente que permite su cálculo. En el siguiente apartado se muestra este algoritmo, cuyo programa se adjunta en el *software* de testeo de secuencias pseudoaleatorias, que se incluye junto a este libro.

6.1.4 El algoritmo de Massey-berlekamp para el cálculo del LCP

Massey abordó [MAS76] el problema de encontrar el registro de desplazamiento con realimentación lineal más corto que generase una secuencia binaria dada, desarrollando un algoritmo que obtenía una solución recursiva para este problema. Este algoritmo sintetiza, para la secuencia $\{s_n\}$ $n = 1, 2,...$, el registro de desplazamiento con realimentación lineal más corto que pueden generar los n primeros dígitos de dicha secuencia.

Un registro de desplazamiento con realimentación lineal (LFSR) consiste en una cascada de L celdas o etapas, las cuales pueden o no servir de entradas a funciones OR-exclusivas para formar una combinación lineal de su contenido que sirve de entrada -realimentada- para la primera celda, tal como muestra la figura 6.2. El contenido inicial de las L celdas, $s_0, s_1,..., s_{L-1}$, coincidirá con los L primeros dígitos de salida, y los restantes bits vendrán determinados de forma única por la recursión:

$$s_j = -\sum_{i=1}^{L} c_i s_{j-i} \qquad j = L, L+1,... \tag{1}$$

Los dígitos de salida y los coeficientes de realimentación $c_1, c_2,..., c_L$, pertenecen al mismo campo, que puede ser un campo finito GF(q) o un campo infinito, como por ejemplo el de los

números reales. Aquí sólo vamos a considerar elementos que pertenezcan al campo de Galois GF(2). Además, no es un requerimiento que $c_L \neq 0$. Se dice que un LFSR genera una secuencia finita c_1, c_2,..., s_{N-1} cuando esta secuencia coincide con los N primeros dígitos de salida del LFSR para una determinada carga inicial. Si $L \geq N$, el LFSR siempre será capaz de generar la secuencia. Si $L < N$, se deduce de (1) que el LFSR generará la secuencia si, y sólo si:

$$s_i + \sum_{i=1}^{L} c_i s_{j-1} = 0 \qquad j = L, L+1, \ldots, N-1 \tag{2}$$

A continuación se muestran una serie de teoremas que llevarán a poder formalizar un algoritmo que permita calcular la complejidad lineal de una secuencia.

TEOREMA 1**:** Si un LFSR de longitud L genera la secuencia s_o, s_1,..., s_{N-1}, pero no es capaz de generar la secuencia s_o, s_1,..., s_N, entonces cualquier LFSR que genere esta última secuencia tendrá una longitud L' que satisface la condición:

$$L' \geq N + 1 - L$$

Sea S una secuencia infinita s_0, s_1,... de forma que s_0, s_1,..., s_{N-1} sean los N primeros dígitos de S. Se define $L_N(s)$ como la mínima longitud de todos los LFSR que generan la secuencia s_o, s_1,..., s_{N-1}. Según lo visto anteriormente, $L_N(s) \leq N$. Pero además, $L_N(s)$ debe ser monótonamente no-decreciente a medida que N crece. Por convenio, se dice que la secuencia todo ceros la generará un LFSR de longitud $L = 0$, y por tanto, $L_N(s) = 0$ si, y sólo si, s_0, s_1,..., s_{N-1} son todos cero.

LEMA 1: Si un LFSR de longitud $L_N(s)$ genera s_o, s_1,..., s_{N-1}, pero no genera la secuencia s_o, s_1,..., s_N, entonces:

$$L_{N+1}(s) \geq \max[L_N(s), N+1-L_N(s)]$$

De la propiedad de monotonía se tiene que $L_{N+1}(s) \geq L_N(s)$. Bajo la hipótesis de este lema, el teorema 1 implica que:

$$L_{N+1}(s) \geq N+1-L_N(s)$$

Veamos ahora cómo se obtiene el algoritmo para producir uno de los LFSR de longitud $L_N(s)$ que generará la secuencia s_0, s_1,..., s_{N-1} para $N = 1, 2, 3,...$ Para ello se define el polinomio de conexión del LFSR de la figura 6.2, $C(D)$, empleando una notación diferente a la que se empleó en el capítulo 3, y que se llama de transformada D:

$$C(D) = 1 + c_1 D + c_2 D_2 + \ldots + c_L D^L$$

que tiene al menos grado L en la indeterminada D. Por convenio, tomamos $C(D) = 1$ para el LFSR de longitud $L = 0$. En el caso binario, los coeficientes s_1, s_2,..., s_i,..., s_L, valen 0 ó 1 según si la conexión de la celda i con la puerta OR-exclusiva (XOR) existe o no.

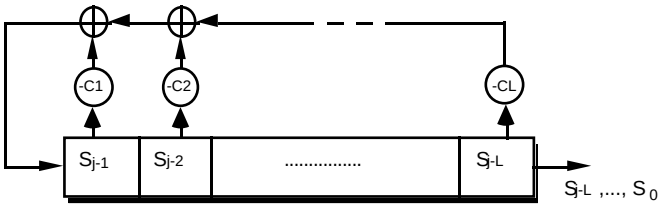


Fig. 6.2 LFSR de L etapas.

Cuando $s_0, s_1,..., s_{N-1}$ son todo ceros, pero $s_N \neq 0$, entonces $L_{N+1}(s) = N+1$, ya que cualquier LFSR más corto se debe cargar inicialmente con todo ceros y sólo podrá generar ceros. Además, en este caso, cualquier LFSR con $L = N+1$ bastará para generar la secuencia $s_0, s_1,..., s_{N-1}, s_N$. Para un s dado, sea

$$C^{(N)}(D) = 1 + c_1^{(N)}D + \ldots + c_{L_{N(s)}}^{(N)} D^{L_{N(s)}}$$

el polinomio de conexión de un LFSR de longitud mínima $L_N(s)$ que genera $s_0, s_1,..., s_{N-1}$. Como hipótesis de inducción, se asume que se ha encontrado $L_N(s)$ y algunos $C^{(N)}(D)$ para $N = 1, 2,...n$. Se pretende entonces encontrar $L_{n+1}(s)$ y algunos $C^{(n+1)}(D)$. Por inducción se tiene a partir de (2) que:

$$S_j + \sum_{i=1}^{L_n(s)} c_i^{(n)} s_{j-i} = \begin{cases} 0 & j = L_n(s),\ldots,n-1 \\ \\ d_n & j = n \end{cases}$$

donde d_n se llamará la *próxima discrepancia* y es la diferencia entre s_n y el $(n+1)$-ésimo dígito generado por el LFSR de mínima longitud que se usa para generar los primeros n dígitos de S. Si $d_n = 0$, entonces este LFSR también generará los primeros $n+1$ dígitos de S, y entonces $L_{n+1}(s) = L_n(s)$, y en este caso deberá tomar $C^{(n+1)}(D) = C^{(n)}(D)$.

Si $d_n \neq 0$, se debe usar un nuevo LFSR para generar los $n+1$ primeros dígitos de S. En este caso, m será la longitud de la secuencia antes del último cambio de longitud en los registros de mínima longitud, es decir:

$$\begin{aligned} L_m(s) &< L_n(s) \\ \\ L_{m+1}(s) &= L_n(s) \end{aligned} \qquad (3)$$

Puesto que se ha necesitado un cambio de longitud, el LFSR con polinomio de conexión $C^{(m)}(D)$ y longitud $L_m(s)$ no puede haber generado la secuencia $s_0, s_1,..., s_{m-1}, s_m$. Entonces, según (3) se tiene que:

$$S_j + \sum_{i=1}^{L_m(s)} c_i^{(m)} s_{j-i} = \begin{cases} 0 & j = L_m(s),\ldots,m-1 \\ \\ d_n \neq 0 & j = m \end{cases}$$

Por la hipótesis de inducción, el lema 1 se cumple con igualdad para $N = m$ para:

$$L_{m+1}(s) = L_n(s) = \max[L_m(s), m+1-L_m(s)]$$

y, en particular, debido a (2), esto da $L_n(s) = m+1-L_m(s)$.

Entonces se propone como candidato válido para actualizar el polinomio de realimentación, es decir, como $C^{(n+1)}(D)$:

$$C(D) = C^{(n)}(D) - d_n d_m^{-1} D^{n-m} C^{(m)}(D)$$

Para el caso binario, las dos discrepancias que aparecen en el polinomio candidato sólo pueden tomar valores 0 ó 1, de manera que cuando ha habido discrepancia podríamos sustituírlas por la constante 1 ($d_n d_m^{-1}$). Así mismo, el signo menos tiene valor de XOR (suma módulo 2) en el caso binario. El grado de $C(D)$ será al menos:

$$\max[L_n(s), n-m+L_m(s)] = \max[L_n(s), n+1-L_n(s)]$$

A partir de ahora, $C(D)$ es un polinomio de conexión admisible para un LFSR de longitud L donde

$$L = \max[L_n(s), n+1-L_n(s)]$$

Entonces, el LFSR de longitud L con polinomio de conexión $C(D)$ genera los $n+1$ dígitos s_0, s_1,..., s_n y se cumple además que $L = L_n(s)$.

TEOREMA 2: Si algún LFSR de longitud $L_n(s)$ que genere la secuencia s_0, s_1,..., s_{n-1}, también genera s_0, s_1,..., s_{n-1}, s_N, entonces $L_{n+1}(s) = L_n(s)$. Por el contrario, si un LFSR que genera la secuencia s_0, s_1,..., s_{n-1} falla al intentar generar la secuencia s_0, s_1, ..., s_{n-1}, s_N, entonces $L_{n+1}(s) = \max[L_n(s), N+1-L_n(s)]$.

Una vez vistos estos teoremas ya se puede ver el algoritmo de Massey-Berlekamp [MAS76] para el cálculo de la complejidad lineal que básicamente sigue los pasos que se han ido viendo:

Paso 1 Inicialización de variables:

$$C(D) := 1 \quad B(D) := 1 \quad x := 1 \quad L := 1 \quad b := 1 \quad N := 1$$

Paso 2 Si $n = N$ parar. Si no, calcular la discrepancia:

$$d = s_N + \sum_{i=1}^{L} c_i s_{j-1}$$

Paso 3 Si $d = 0$ entonces $x = x+1$, e ir a paso 6

Paso 4 Si $d \neq 0$ y $2L > N$, entonces:

$C(D) = C(D) - db^{-1}D^xB(D)$ (se actualiza $C(D)$)
$x := x+1$
ir al paso 6

Paso 5 Si $d \neq 0$ y $2L \leq N$ entonces:

$T(D) = C(D)$ (se guarda temporalmente $C(D)$ en $T(D)$)
$C(D) = C(D) - db^{-1}D^xB(D)$ (se actualiza $C(D)$)
$L := N+1-L$ (se actualiza L)
$B(D) = T(D)$ (se actualiza $B(D)$)
$b := d$
$x := 1$

Paso 6 $N := N+1$ y volver al paso 2.

Para todo n, cuando $N = n$ y se ha alcanzado el paso 2, las cantidades producidas por el algoritmo tienen las siguientes relaciones:

$C(D) = C^{(n)}(D)$
$L = L_n(s)$
$x = n - m$
$d = d_n$ (suponiendo que el cálculo en el paso 2 se ha realizado)
$B(D) = C^{(m)}(D)$
$b = d_m$

Notar que el paso 5 sólo se realiza cuando, de acuerdo con el teorema 2, se necesita un cambio de signo. Sólo se ha considerado el problema de encontrar uno de los registros de mínima longitud que puede generar una secuencia dada, pero realmente se podría encontrar todo el conjunto de LFSRs de longitud mínima $L_n(s)$ que puede generar la secuencia $s_0, s_1,..., s_{n-1}$ mediante este algoritmo de síntesis. También se puede concluir que el LFSR de mínima longitud es único si, y sólo si, $2L_N(s) \leq N$. En el *software* que se adjunta se puede encontrar un programa que implementa este algoritmo.

A) Relación entre la complejidad lineal y la de Turing-Kolmogorov-Chaitin

En este apartado se analiza la relación entre la complejidad lineal $\Lambda(\underline{s})$ y la complejidad de Turing-Kolmogorov-Chaitin $\chi(\underline{s})$. El siguiente teorema establecerá esta relación:

TEOREMA 1: Para cualquier número real ε que cumple que $0 < \varepsilon < 1$, un entero arbitrario n y una secuencia binaria arbitraria $\underline{s} \in 2^n$, se cumplirá que:

$$1)\ Prob\{(1-\varepsilon)\Lambda(\underline{s}) \le \chi(\underline{s})\} \ge 1 - \frac{8}{3}2^{-\frac{\varepsilon}{2-\varepsilon}n}$$

$$2)\ Prob\{(1-\varepsilon)\chi(\underline{s}) \le \Lambda(\underline{s})\} \ge 1 - \frac{1}{3}2^{-\varepsilon n+(1-\varepsilon)(1+\log n)+1} - \frac{1}{3}2^{-n}$$

De este teorema se puede deducir el siguiente corolario.

COROLARIO: Para todo ε, tal que $0 < \varepsilon < 1$, se cumplirá que:

$$P_{\varepsilon,n} = Prob\{\underline{s} \in 2^n \,|\, (1-\varepsilon)\cdot 2\Lambda(\underline{s}) \le \chi(\underline{s}) \le (1+\varepsilon)\cdot 2\Lambda(\underline{s})\} \to 1$$

cuando $n \to \infty$.

A modo de ejemplo, para ε = 0,01 tendremos que para secuencias de longitud n:

n	2000	4000
$P_{\varepsilon,n}$	0,998	0,999997

6.1.5 La complejidad de Ziv-Lempel

Otra medida de complejidad de las secuencias la definieron Ziv y Lempel en 1976 [LEM76]. Esta complejidad mide la velocidad a la que emergen nuevas cadenas de bits a medida que nos movemos a través de la secuencia. Hasta ahora la complejidad de Ziv-Lempel se había usado principalmente en conexión con el algoritmo de Ziv-Lempel para la compresión de datos. En el estudio de secuencias para su aplicación en criptografía sólo había sido aplicada por Leung y Tavares [LEU85] para testear cifradores de bloque, pero no se había usado para testear cifradores de flujo (este tipo de criptosistemas se verán en el capítulo 7).

El cálculo de esta complejidad se puede ver desde el punto de vista de una máquina de aprendizaje autolimitada que, a medida que estudia la secuencia de n dígitos S = s_1, s_2,..., s_n de izquierda a derecha, añade una nueva palabra a su memoria cada vez que descubre una subsecuencia de dígitos consecutivos que no había aparecido previamente. El tamaño del vocabulario compilado y la velocidad a la que van apareciendo a lo largo de la secuencia *S* sirven como elementos básicos para el cálculo de la complejidad de *S*, es decir, la complejidad está relacionada con la velocidad a la que crece el vocabulario. Esto hará que no se puedan considerar como secuencias de elevada complejidad aquellas secuencias que presenten regularidades estructurales o vocabularios deficientes.

Por tanto, intuitivamente la complejidad de Ziv-Lempel servirá como medida de las secuencias finitas, basándose esta medida en la velocidad a la que emergen nuevos patrones a medida que nos movemos por la secuencia.

Sea S = s_1, s_2,..., s_n una secuencia de longitud n. El siguiente procedimiento [MUNDS] muestra la forma para calcular la complejidad de Ziv-Lempel $Z(\underline{s})$ de la secuencia S:

Paso 1	Se introduce una barra oblicua separadora despues de s_1.
Paso 2	Asumimos que la i-ésima barra separadora viene después del dígito s_{k_i}, donde $1 \le k_i \le n-1$. La siguiente barra se insertará después del dígito $s_{k_{i+1}}$, donde $k_{i+1} = k_i + L_i + 1 \le n$, y L_i es la máxima longitud de una subsecuencia $s_{k_i+1}....s_{k_i+L_i}$, tal que existe un entero p_i (donde $1 \le p_i \le k_i$) para el cual $s_{p_i}...s_{p_i+L_i} = s_{k_i+1}...s_{k_i+L_i}$.

Si s_n está seguida por una barra, la complejidad de Ziv-Lempel será igual al número de barras mientras que en otro caso, la complejidad de Ziv-Lempel será igual al número de barras más uno. Veamos un ejemplo para ilustrar esto.

EJEMPLO 6.1: Sea la secuencia binaria S = 1001101110000111. Si le introducimos barras siguiendo la metodología introducida por los dos pasos anteriores obtendremos la partición S = 1/0/01/101/1100/00111. Esta partición divide a la secuencia en 6 patrones distintos, y por tanto la complejidad de Ziv-Lempel de dicha secuencia será de 6.

Sin embargo, el paso 2 del procedimiento anterior no es fácilmente computable por lo que es conveniente proponer una forma eficiente de computarlo. Para ello se puede subdividir el paso 2 en otros dos:

Paso 2a	Inicialización: Sea n+1 la posición de la secuencia donde comienza el cómputo de un nuevo patrón, sea J, tal que contenga todas las posiciones j para las cuales $s_j = s_{n+1}$ ($1 \le j \le n$) y sea m = 2.
Paso 2b	Repetir este paso hasta que J esté vacío: Eliminar todo j $\in J$ para el cual $s_{j+m-1} \neq s_{n+1}$, e incrementar m en 1 si J todavía no está vacío. En otro caso, m será la longitud del nuevo patrón que vendrá definido por $s_{n+1}...s_{n+m}$.

Estos dos pasos nos permitirán obtener los patrones de la secuencia S de forma eficiente y, por tanto, calcular la complejidad de Ziv-Lempel de dicha secuencia. El algoritmo de Ziv-Lempel se centra, pues, en una tabla en la que se ponen las nuevas palabras que van apareciendo. Normalmente se usan códigos de 12 bits para la tabla (lo que permite 4096 entradas).

Veamos algunos lemas que nos resultarán útiles para calcular la complejidad de Ziv-Lempel de secuencias periódicas s_1,..., s_{p-1}, s_p, s_{p+1},..., donde p es el periodo de la secuencia, y $s_{p+i} = s_i$ para $0 < i \le p$.

LEMA 1: Si S es una secuencia periódica donde $s_1...s_p$ tiene complejidad de Ziv-Lempel $Z(\underline{s})$, y $s_1...s_{p+1}$ tiene complejidad de Ziv-Lempel $Z(\underline{s})+1$, entonces para $m \geq q = p+1$, la secuencia $s_1...s_m$ tendrá complejidad de Ziv-Lempel $Z(\underline{s})+1$.

LEMA 2: Si S es una secuencia de periodo p, complejidad de Ziv-Lempel $Z(\underline{s})$ para $s_1...s_{p-k}$, y complejidad de Ziv-Lempel $Z(\underline{s})+1$ para $s_1...s_{p+1}$, entonces existe un q tal que la secuencia $s_1...s_j$ tiene una complejidad de Ziv-Lempel $Z(\underline{s})+1$ para $p-k < j \leq q$, y una complejidad de Ziv-Lempel $Z(\underline{s})+2$ para $j > q$ ($k > 0$).

LEMA 3: A partir del lema 2, la máxima longitud del modelo $s_{p-k+1}...s_{p+1}$ será $p-k$.

LEMA 4: $q \leq 2p-2$ bajo la suposición del lema 3.

LEMA 5: Teniendo en cuenta la suposición del lema 3 para k fija y $m \leq p-k$ fija, j debe ser elegida de tal forma que $j+1 \leq p-k$. Con $2p-1$ bits de la secuencia periódica será suficiente para calcular la complejidad de Ziv-Lempel.

La complejidad de Ziv-Lempel de secuencias periódicas depende del punto en que se empiece a analizar y, por tanto, podemos obtener diversas complejidades según el punto de comienzo.

LEMA 6: Sea S una secuencia periódica con periodo p, $Z_1(\underline{s})$ la complejidad de Ziv-Lempel de la secuencia calculada comenzando en la posición s_i, y sea $Z_2(\underline{s})$ la complejidad de Ziv-Lempel calculada comenzando en la posición s_{i+1}, entonces $Z_1(\underline{s})$ y $Z_2(\underline{s})$ pueden ser diferentes.

La complejidad de Ziv-Lempel se puede usar para examinar secuencias pseudoaleatorias. En adelante se asumirá que $p(s_j=0) = p(s_j=1) = 0{,}5$. En primer lugar se examina la longitud media de los patrones que comienzan en la posición s_{n+1}.

LEMA 7: La longitud media de los patrones que comienzan en la posición s_{n+1} es $p_a = \lfloor log_2 n \rfloor + 1$, donde $\lfloor x \rfloor$ indica el valor entero del número x.

También se puede hacer una estimación de la complejidad media de Ziv-Lempel $\overline{Z}(\underline{s})$ de una secuencia de longitud n suponiendo que cada patrón de la secuencia se calcula independientemente de los patrones anteriores. La estimación se puede obtener mediante las relaciones de recurrencia:

1) Para $j = 1$ $\overline{Z}(\underline{s}) = 1$ y $j_0 = 1$.
2) Para $j = 2...n$: Si $j = j_0 + \lfloor log_2 j_0 \rfloor + 1$ entonces $\overline{Z}(\underline{s}) = \overline{Z}(\underline{s}) + 1$ y $j_0 = j$.

Las dos medidas vistas, la complejidad media de Ziv-Lempel $\overline{Z}(\underline{s})$ y la longitud de los modelos $\overline{p}$ se pueden usar para examinar secuencias de números pseudoaleatorios.

Se espera que una buena secuencia aleatoria tenga una complejidad de Ziv-Lempel cercana a $\overline{Z}(\underline{s})$, y que cada modelo de la secuencia tenga una longitud p_i cercana a $\overline{p}$. Si la complejidad de Ziv-Lempel de la secuencia es mucho menor que $\overline{Z}(\underline{s})$, la secuencia tendrá al menos una repetición múltiple de un patrón y existirá al menos una longitud de patrón p_i que es mucho mayor que $\overline{p}$. El modelo *i* será una repetición de patrones aparecidos ya anteriormente. La repetición múltiple de un patrón será una desventaja si el generador de secuencias pseudoaleatorias se usa en una aplicación como el cifrado en flujo. Si la secuencia pseudoaleatoria no sólo tiene una repetición múltiple de un patrón, sino que el texto en claro también lo tiene, el texto cifrado correspondiente debería tener también una repetición múltiple de patrones. Por tanto, la complejidad de Ziv-Lempel y la longitud de los patrones se pueden usar para determinar si una secuencia de números pseudoaleatorios es deseable o no.

A) Relación entre la complejidad lineal y la complejidad de Ziv-Lempel

Al estudiar la complejidad de una secuencia podemos encontrarnos con que ésta muestre una elevada complejidad lineal pero, sin embargo, al estudiar su complejidad de Ziv-Lempel nos dé un valor muy bajo. Un ejemplo de ello podría ser el caso de la secuencia S = 0000....000001 que se vio en el apartado 6.1.2. En este apartado también se vio que la complejidad lineal no dice mucho por sí sola sobre la calidad de la secuencia. Si consideramos la complejidad de Ziv-Lempel de una secuencia de este tipo, se ve inmediatamente que da un valor mínimo de complejidad, con lo que nos indica que la secuencia no será adecuada para aquellas aplicaciones que requieran esta propiedad. Se puede observar que, en este caso, la baja complejidad de Ziv-Lempel de esta secuencia nos da la misma información que en su momento nos había revelado el calcular el perfil de complejidad lineal de la secuencia. Es decir, tenemos una secuencia con complejidad lineal máxima pero no será conveniente utilizarla en ciertas aplicaciones, ya que el perfil de complejidad lineal no crece gradualmente sino de forma brusca y presenta una complejidad de Ziv-Lempel mínima. Esto nos lleva a considerar la complejidad de Ziv-Lempel como una medida adicional sobre la calidad de la secuencia que nos dará más información sobre la repetición de patrones a lo largo de ésta. En particular, esta complejidad puede utilizarse para detectar repeticiones de patrones que no afecten a la complejidad lineal y, por tanto, serán detectables por ésta.

6.1.6 La complejidad de máximo orden

Una nueva medida de la complejidad de una secuencia es la que Jansen y Boekee definen como la complejidad de máximo orden [JAN89][JAN91]. Esta complejidad se define como la longitud del registro de desplazamiento más corto que puede generar la secuencia teniendo en cuenta que su función de realimentación puede ser cualquiera. El nombre de máximo orden viene del hecho que, a diferencia de la complejidad lineal (de primer orden), la complejidad cuadrática (de segundo orden), etc, en este caso no existe restricción en el orden de la función de realimentación. Veamos algunas definiciones acerca de esta complejidad.

TEOREMA 1: La complejidad de máximo orden $C(\underline{s})$ de una secuencia $S = s_1, s_2, \ldots s_n$, con caracteres $s_i \in A$, donde A es un alfabeto finito, se define como la longitud L del registro de desplazamiento

realimentado más corto para el cual existe una función de transformación sin memoria, tal que usada como realimentación de dicho registro, éste sea capaz de producir la secuencia *S*.

Veamos algunas propiedades asociadas a la complejidad de máximo orden:

PROPIEDAD 1: Para una secuencia *S* consistente en 2 o más caracteres diferentes y posiblemente repetidos, la complejidad $C(\underline{s})$ es igual a la longitud de la mayor subsecuencia que ocurra al menos dos veces con el carácter siguiente diferente más uno.

PROPIEDAD 2: La complejidad de una secuencia es 0 si dicha secuencia consiste en un solo posible carácter que se repite.

PROPIEDAD 3: El máximo valor de la complejidad de máximo orden de una secuencia de longitud *n* será *n*-1. Una secuencia de longitud *n* tendrá una complejidad de *n*-1 si consiste en *n*-1 copias consecutivas de algún determinado carácter, seguido por otro carácter distinto.

Para secuencias periódicas de periodo *p*, la complejidad de máximo orden presentará las siguientes propiedades:

PROPIEDAD 4: La mínima complejidad de una secuencia periódica de periodo *p* es $\lfloor log_a\ p \rfloor$, donde $a = |A|$ es la cardinalidad del alfabeto de carácteres, y $\lfloor x \rfloor$ representa la parte entera del número *x*.

PROPIEDAD 5: La complejidad máxima de una secuencia de periodo *p* es *p*-1.

Al igual que ocurría con la complejidad lineal, el hecho de que la secuencia estudiada presente una elevada complejidad de máximo orden no es condición suficiente para demostrar la calidad de la secuencia, y también es necesario estudiar cómo se produce el crecimiento de esta complejidad al estudiar esta secuencia.

De la definición que se ha hecho de la complejidad de máximo orden, es evidente que en el caso de que la función de realimentación sea de primer orden coincidirá con la complejidad lineal. Esta situación se dará con las secuencias generadas por LFSR o *m*-secuencias.

Veamos un ejemplo:

EJEMPLO 6.2: Consideremos la siguiente secuencia de longitud 25 obtenida con un dado: *S* = 6544552566433434162531433. Esta secuencia presenta una complejidad de 3 caracteres, ya que todas las subsecuencias de longitud 3 son distintas. Sin embargo, la subsecuencia (43) tiene dos sucesores diferentes $(43\underline{3}\ \ y\ \ 43\underline{4})$.

PROPIEDAD 6: Una secuencia periódica $S = (s_1, s_2, ..., s_P)$ y su recíproca $S^* = (s_P, s_{P-1}, ..., s_1)$ tienen la misma complejidad.

Esta propiedad no será válida para secuencias no periódicas ya que, por ejemplo, la secuencia de longitud *n* (aaaa...aaab) tendrá complejidad *n*-1, y su recíproca (baaa...aaa) tendrá complejidad 1.

PROPIEDAD 7: Si llamamos $\Phi_{\underline{S}}$ a la clase de funciones de realimentación del registro de desplazamiento con realimentación de máximo orden equivalente de la secuencia periódica *S* sobre GF(*q*), con *C* y *p* la complejidad y el periodo de la secuencia *S* respectivamente. El número $\left|\Phi_{\underline{S}}\right|$ de funciones en la clase $\Phi_{\underline{S}}$ satisface la ecuación:

$$\left|\Phi_{\underline{s}}\right| = q^{q^C - p}$$

Por tanto, a diferencia de la complejidad lineal en la que la función de realimentación es única para secuencias periódicas, $\Phi_{\underline{S}}$ contendrá en general más de una función y podremos buscar entre ellas algunas que tengan ciertas propiedades como: no-singularidades, el producto de menor orden, o la función que tenga menor número de términos.

En 1986, Blumer propuso un algoritmo eficiente, lineal en tiempo y memoria, para calcular el perfil de complejidad de máximo orden, llamado grafo de palabras directo acíclico (DAWG).

Además, el algoritmo de Blumer se puede emplear para otros propósitos interesantes como calcular el periodo de una secuencia periódica, encontrar una determinada subsecuencia dentro de una secuencia dada, o generar una determinada secuencia basándose en el menor número de caracteres observados.

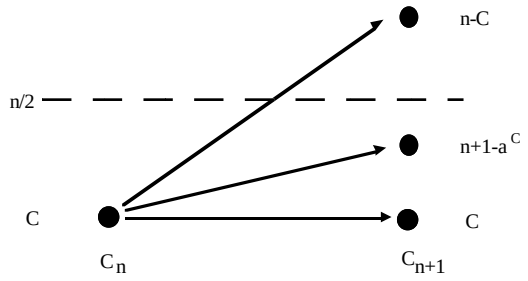


Fig. 6.3 Posibles saltos en el perfil de complejidad de máximo orden

Se puede ver, para finalizar, cuál será el comportamiento del crecimiento de la complejidad de máximo orden, lo que determinará su perfil de complejidad. Supongamos que tenemos una secuencia *S* de longitud *n* cuyos elementos pertenecen a un alfabeto finito *A* y usamos el símbolo C_n para expresar la complejidad de *n* elementos de la secuencia. Al igual que ocurría con la complejidad lineal, la línea *n*/2 constituirá un valor límite que acotará el crecimiento, y determinará si el perfil de complejidad salta hacia un nuevo valor mayor o se mantiene en el mismo. Veamos algunas propiedades relacionadas con este perfil de crecimiento:

PROPIEDAD 8: Si la secuencia *S* de longitud *n* presenta complejidad C_n, entonces el valor de la complejidad C_{n+1} de la secuencia extendida con un nuevo símbolo s_{n+1} vendrá dado por:

$$C_{n+1} = \begin{cases} = C_n & C_n \geq n/2 \\ \leq n - C_n & C_n < n/2 \\ \geq C_n + n & 2C_n < n < C_n + a^{C_n} \\ \geq n + 1 - a^{C_n} & n \geq C_n + a^{C_n} \end{cases}$$

donde $a = |A|$, tal como se definió anteriormente, representa la cardinalidad del alfabeto de caracteres. Esta propiedad muestra exactamente la forma en la que el perfil de complejidad de máximo orden salta de un valor a otro al pasar a un nuevo carácter. Esto se puede ver también en la figura 6.3.

6.2 Problemas

PROBLEMA 6.1 Determinar la complejidad lineal de las siguientes secuencias:

a) S_1 = 0000000000.....
b) S_2 = 0101010101.....
c) S_3 = 00000000000000000000000000001
d) S_4 = 00110100011110101001111001111111000100100

PROBLEMA 6.2 Calcular el crecimiento de la complejidad de la secuencia S = 00101001110110101111001010000010 utilizando el algoritmo de Massey-Berlekamp.

PROBLEMA 6.3 Calcular, para la misma secuencia del apartado anterior, el crecimiento de la media y la varianza de la complejidad lineal.

PROBLEMA 6.4 Determinar la complejidad de Ziv-Lempel para la secuencia S = 10111011010110111000101001110

PROBLEMA 6.5 Determinar la complejidad de Ziv-Lempel para las secuencias del problema 6.1.

6.3 Bibliografía

[ASS74] ASS, J. *Information-theoretic Limitations of Formal Systems*. Comput. Mach., Vol. 21, pp. 403-424, 1974.

[BET89] BETH, T.; DAI, Z. D. *On the Complexity of Pseudo Random Sequences-or: If You Can Describe A Sequences It can´t be Random*. E.I.S.S., Report 89/4, 1989.

[CHA74] CHAITIN, G. *A Theory of Program Size Formally Identical to Information Theory*. IBM, Yorktown Haights, N.Y., Rep, RC 4805, April 1974.

[CHA82] CHAN, A. H.; GAMES, R. A.; KEY, E. L. *On the distribution of De Bruijn Sequences of Given complexity*. J. comb theory, vol. 3, Nº 3, pp. 223-246, Nov. 1982.

[DAI86] DAI, Z. D. *Proof of Ruepple`s Linear Complexity Conjeture*. IEEE Trans. on Inform. theory, Vil. IT-32, pp. 440-443, 1986.

[GOL88] GOLDWASSER, S.; MICALI, S. *The Knowledge Complexity of Interactive Proof Systems*. Siam J. Comput, 17, pp. 281-308, 1988.

[GOL88] GOLIC, J. D.; ZIVKOVIC, M. V. *On the Linear Complexity of Non-uniformly Decimated PN-Sequences*. IEE Trans. on Inform theory, Vol. 34, pp. 1077-1079, Sept. 1988, Part I.

[GOL89] GOLIC, J. D. *On the Linear Complexity of functions of Periodic GF(q) Sequences*. IEE Trans. on inform theory, Vol. 35, nº 1, Jan. 1989.

[GRO71] GROTH, E. J. *Generation of Binary Sequences with Controllable Complexity*. IEEE Trans. on Inform theory, Vol. IT-17, pp. 288-296, May.1971.

[HER82] HERLESTAM, T. *On the Complexity of functions of Linear Shift Register Sequences*. Int. Symp. Infrom. Theory, Les Arcs, France, 1982.

[JAN89] JANSEN, C. J. A.; BOEKEE, D. E. *The Shortest Feedback Shift Register that can Generate a Given Sequence*. Proceedings Crypto 89, Santa Barbara. Proc. Crypto 89. Springer-Verlag Lecture Notes in Computer Science, New York, 1989.

[JAN91] JANSEN, C. J. A. *The Maximum Order Complexity of Sequences Ensembles*. Proc. Eurocrypt´91, 1991.

[KEY76] KEY, E. L. *An Analisys of the Structure and Complexity of Non-Linear Binary Sequences Generator*. IEEE Trans. on Inform. theory. Vol. IT-22, pp. 732-736, Nov. 1976.

[KOL65] KOLMOGOROV, A. N. *Three Approaches to the Quantitative Definition of Information*. Probl. Inform. Transmission, Vol 1. pp.1-7, 1965.

[KWOKR] KWOK, R. T. C.; BEALE, M. *A periodic Linear Complexities of De Bruijn Sequences*. Univ. of Manchester, England.

[MAR66] MARTIN-LÖF, P. *The Definition of Random Sequences*. Inf. Contr., Vol. 9, pp. 60, 1966.

[MAS76] MASSEY, J. L. *Shift Register Sysnthesis and BCH Decoding*. IEEE Trans on Information Theory, Vol. IT-15, No 1, Jan. 1976.

[MUNDS] MUND, S. *Ziv-Lempel Complexity for Periodic Sequences and its Cryptographic Application*. Siemens AG, München, West Germany.

[LEM76] LEMPEL, A.; ZIV, J. *On the Complexity of Finite Sequences*. IEEE Transactions on Inform. Theory, Vol IT-22, Nº1, Jan. 1976.

[LEU85] LEUNG, A.; TAVARES, S. *Sequence Complexity as a Test for Cryptographyc Systems*. Advances in Cryptology. Crypto 84, Lecture Notes in Computer Science, Springer-Verlag, Heidelberg, 1985.

[RUE86] RUEPPEL, R. A. *Linear Complexity and Random Sequences*. Proceeding of Eurocrypt´86, Lecture Notes in computer Science 219, Springer-Verlag, 1986.

[RUE87] RUEPPLE, R. A.; STAFFELBACH, O. *Products of Linear Recurring Sequences with Maximum Complexity*. IEEE Trans. on Inform. theory, Vol. IT-33, Nº 1, Jan. 1987.

[SCH73] SCHNORR, C. P. *The Process Complexity and Effective Random Tests*. J.Comput. Syst. Sci., Vol 7, pp.376-388, 1973.

[WAH71] WAHRSCHEINLICHKEIT, Z. *Complexity Oscillations in Infinite Binary Sequences*. Z. Wahrscheinlichkeit verw. Geb., Vol 19, pp. 225-230, 1971.

[YANGJ] YANG, J. H.; DAI, Z. D. *Linear Complexity of Periodically Repeated Sequences*. Academia Sinica, Beijing, China.

[ZIV78] ZIV, J. *Coding Theorems for Individual Sequences*. IEEE Trans. on Inform. Theory, Vol. IT-24, Nº 4, 1978.

7 Aplicaciones de las secuencias pseudoaleatorias

7.0 Introducción

Ya se ha visto en capítulos anteriores las diferentes técnicas que permiten generar secuencias de números pseudoaleatorios y cómo comprobar sus propiedades. Tal como se vio en el capítulo 1, las propiedades que se van a requerir en este tipo de secuencias no van a ser siempre las mismas, sino que dependerán de la aplicación en la que vayan a ser utilizadas. Por ejemplo, la propiedad de elevada complejidad es una característica esencial en aplicaciones de cifrado en flujo pero, sin embargo, el hecho de que no se cumpla no afecta en absoluto el buen funcionamiento de estas secuencias en otras aplicaciones, como por ejemplo las comunicaciones *spread spectrum*, la dispersión de energía de señales en comunicaciones o los sistemas de acceso múltiple por división de código, que también utilizan este tipo de secuencias. En este capítulo se van a revisar algunas de las aplicaciones que requieren del uso de secuencias pseudoaleatorias para su funcionamiento, analizando las propiedades que deberán cumplir estas en cada caso concreto y algunos ejemplos de sistemas prácticos.

7.1 El cifrado en flujo

Si bien hasta hace poco la criptología [BEK89][MAT88] era un campo de uso reservado casi exclusivamente a diplomáticos y militares, la llegada de la era de la información, con sus avances tanto sociales como tecnológicos, ha traído consigo nuevas necesidades como es la de garantizar la privacidad de las comunicaciones. Además, la enorme potencia de cálculo que permiten los procesadores actuales ha hecho de esta ciencia un área de gran interés, principalmente por el abanico de aplicaciones, cada vez mayor, que puede soportar. Uno de los campos de interés dentro de la criptografía es el del cifrado en flujo [PIP82]. En este tipo de cifrado, la protección se basa en combinar la información a transmitir con la salida producida por un generador de secuencias pseudoaleatorias, de forma que la seguridad final ofrecida dependerá de las propiedades de éste. Esta aplicación es la que ha propiciado la gran proliferación de generadores de secuencias pseudoaleatorias que se ha producido en los últimos años, la mayoría de los cuales se han visto en el capítulo 4.

7.1.1 Fundamentos de criptografía

La criptografía es la ciencia que trata de la protección de la información para evitar el acceso a ella por parte de personas no autorizadas [SHA49][BEK82]. Para ello, deben realizarse un conjunto de

transformaciones T_k a un mensaje o texto en claro m (que pertenece a un espacio de mensajes M) que se quiere transmitir de forma segura a través de un canal que se supone interceptado, obteniendo de esta forma un texto cifrado $c = T_k(m)$ también llamado criptograma (y que pertenece a un espacio de claves C). Este criptograma sólo podrá ser descifrado [SHA49] por el receptor legítimo mediante la aplicación de la transformación inversa T_k^{-1}, para obtener de ese modo el texto original m:

$$T_k^{-1}(c) = T_k^{-1}(T_k(m)) = m$$

El conjunto de reglas de transformación $M \Leftrightarrow C$ se denomina algoritmo criptográfico [BEK89] y depende de un parámetro k que se denomina clave, que selecciona qué transformación del conjunto se va a utilizar.

En aplicaciones prácticas, se puede suponer que el algoritmo de cifrado es conocido, por lo que la seguridad del sistema recaerá en el desconocimiento de la clave por parte de un posible atacante, ya que si éste no la conoce deberá probar, de manera exhaustiva, todas las posibles combinaciones. A este tipo de ataques se les suele denominar por fuerza bruta.

Más adelante se verán diversos métodos que permitirán a un atacante reducir el tiempo que necesitaría para descubrir la clave utilizada mediante la búsqueda exhaustiva y poder romper así el criptosistema. Dichos métodos se basan en encontrar distintas redundancias en las secuencias pseudoaleatorias utilizadas para realizar el cifrado. Esto ya nos permite intuir que la calidad de estas secuencias será un factor crítico en la seguridad.

Tal como se ha mencionado, la idea subyacente de la criptografía moderna es que los algoritmos criptográficos sean públicos, de forma que su seguridad no esté condicionada al hecho del desconocimiento de éste por parte de posibles atacantes. Sin embargo, en la práctica, la mayoría de los sistemas actuales mantienen los algoritmos de cifrado y descifrado de forma secreta como refuerzo adicional a la seguridad.

Un sistema criptográfico debe intentar cubrir una serie de necesidades resultantes de la aplicación de un objetivo de seguridad para la información [BOY89][CAM78]:

1) Evitar que el mensaje sea descubierto (confidencialidad).

2) Evitar que el mensaje pueda ser modificado de forma selectiva.

3) Evitar la inserción de mensajes falsos.

4) Detectar modificaciones del texto cifrado.

Para que un sistema sea de secreto perfecto, el número de claves (K) debe ser, al menos, igual al número de mensajes con probabilidad no nula. Como en la práctica no es posible generar una clave de longitud infinita, nos tendremos que conformar con obtener una secuencia pseudoaleatoria de periodo mucho mayor que la longitud del mensaje que se quiera proteger. Para ello se pueden emplear tanto técnicas de transposición (reordenación de los caracteres del texto original) como de sustitución (basadas en combinar el texto que se quiere transmitir con las secuencias producidas por generadores de

secuencias pseudoaleatorias). En los siguientes apartados nos centraremos en sistemas de cifrado basados en esta última técnica, que para ser efectivos deberán cumplir una serie de requisitos [BEK82]:

1) Las transformaciones que realice el sistema han de ser reversibles, para todo mensaje producido por la fuente y para toda clave que pertenezca al espacio de claves:

$$T_k^{-1}(T_k(m)) = m \qquad \text{donde m} \in M \text{ y } k \in K$$

2) La fortaleza del sistema debe basarse en la dificultad de obtener la clave a partir del propio sistema y del texto fuente, y no en el desconocimiento de la familia de transformaciones.

3) Para dos familias de transformaciones análogas, tendrá mayor seguridad aquella que tenga un espacio de claves (K) mayor.

4) El sistema de distribución de claves ha de ser suficientemente seguro, ya que la seguridad introducida por la trasformación está condicionada por el desconocimiento de la clave.

Además, al tratar un sistema de estas características se deben hacer una serie de hipótesis de trabajo previas, como el hecho de que el criptoanalista tiene un conocimiento completo del sistema de cifrado, ha conseguido una cantidad considerable de texto cifrado y conoce el texto en claro correspondiente a una cierta porción del texto cifrado. Si bien estas tres suposiciones pueden parecer pesimistas, son casi siempre realistas, y cualquier sistema de cifrado debe ser seguro bajo las mismas. Por supuesto, para cada situación, los términos considerable y cierta porción deben ser cuantificados. Sus valores precisos dependerán del sistema que se considere y el nivel de seguridad requerido. Este nivel de seguridad vendrá determinado por el cumplimiento de una serie de condiciones:

1) Para usuarios autorizados, las operaciones de cifrado y descifrado deben ser simples y eficaces.

2) La operación de descifrado deberá ser muy complicada para los usuarios no autorizados.

3) No debe ser una condición imprescindible que el algoritmo de cifrado se mantenga en secreto.

4) La eficiencia y seguridad obtenida debe ser independiente de los datos que se vayan a transmitir.

La división más común entre los cifradores es la que los clasifica en cifradores de bloque o cifradores de flujo. Mientras que los cifradores en bloque dividen el texto en bloques de tamaño fijo, y operan de forma independiente sobre cada uno de ellos (son pues cifradores por sustitución simple y con un solo estado interno), los cifradores en flujo dividen el texto en caracteres y cifran cada unidad del mensaje (por ejemplo cada bit) mediante una función, cuya dependencia con el tiempo viene gobernada por el estado interno del cifrador, de forma que tras el cifrado de cada carácter, el dispositivo

cambia de estado interno de acuerdo a una cierta regla. Esto hace que en un cifrador en flujo, dos caracteres iguales en el texto a cifrar puedan dar dos caracteres diferentes en el texto cifrado, a diferencia de lo que ocurre en los de bloque. Los cifradores en flujo pueden poseer varios estados internos y requieren de un vector de inicialización que será la clave del criptosistema. En este capítulo sólo se estudiarán los cifradores de flujo que, como se verá más adelante y por su forma de funcionamiento, necesitan disponer de una secuencia pseudoaleatoria generada por algún método como los que se han visto en los capítulos anteriores para realizar el cifrado.

7.1.2 Tecnología de cifradores de flujo

Los cifradores de flujo [PIP82][BEK82] intentan simular el sistema de cifrado *one-time-pad* que propuso G. Vernan en 1917 [VER26]. Tal como se ve en la figura 7.1, este sistema se basa en el uso de una secuencia de bits aleatoria de clave, que se genera por el procedimiento de tirar monedas de forma sucesiva e independiente (esto es equivalente a usar una fuente binaria simétrica o BSS que genere dicha secuencia).

Posteriormente, esta secuencia (secuencia cifrante) se suma (módulo 2) bit a bit con el texto en claro para producir el texto cifrado. Un ejemplo de cómo obtener el texto cifrado es:

Texto en claro *m*:	011000111111101.....
Clave secreta *k*:	100110010001011.....
Texto cifrado *c*:	111110101110110.....

En este sistema, la clave debe ser tan larga como el mensaje y el conocimiento de parte de ella no debe dar información sobre ninguna otra parte.

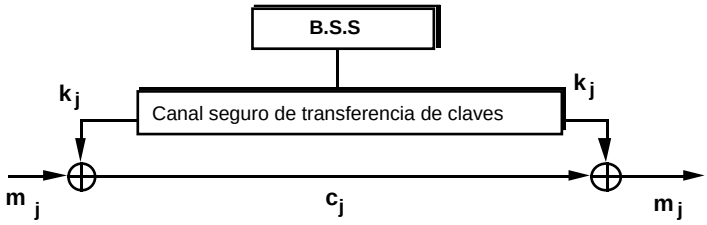


Fig. 7.1 Cifrador de Vernan

Si la clave *k*, que es precisamente la secuencia aleatoria, está realmente producida por una fuente binaria simétrica, es decir, por una fuente capaz de producir en cada instante un uno o un cero con probabilidd 1/2 independientemente de los bits producidos anteriormente, se puede considerar al sistema *one-time-pad* como un sistema completamente seguro contra cualquier tipo de ataque que parta tan sólo del conocimiento del texto cifrado y no disponga de información del mensaje que se cifró. Se puede observar que si el atacante dispusiera además de parte del texto en claro (lo que puede darse con

mucha probabilidad en la práctica), podría obtener, a partir de ambos, parte de la secuencia aleatoria. Esto no será un problema serio si la secuencia es realmente aleatoria. Sin embargo, si ésta está generada a partir de máquinas de estados finitos, tales como los que se han visto en capítulos anteriores, puede representar un problema si la secuencia de dicho generador muestra una elevada predictibilidad, ya que un ataque criptoanalítico permitiría obtener la estructura que la produce y entonces el atacante podría, al disponer ya de la secuencia pseudoaleatoria completa, descifrar totalmente todo el texto cifrado. Más adelante se verán algunos de los métodos criptoanalíticos más utilizados.

Aunque el sistema *one-time-pad* se considera, al menos teóricamente, incondicionalmente seguro debido a que presupone utilizar secuencias completamente aleatorias, presenta problemas de difícil solución en la práctica a la hora de generar y distribuir las claves. Esto es debido a que, puesto que el flujo de bits de clave debe ser tan largo como el mensaje a cifrar, y debe ser totalmente aleatorio, la única manera posible de que el receptor use la misma secuencia que el transmisor es enviándosela (¡de forma segura!) antes de enviarle el criptograma o mensaje cifrado, para que pueda descifrarlo.

Los cifradores de flujo intentan solucionar el problema de la gestión (distribución) de claves usando un algoritmo determinista y una clave finita para generar un flujo de bits aleatorios que se sumará, bit a bit, al texto en claro para cifrarlo, tal como se muestra en la figura 7.2. La clave finita (y secreta) *k* debe tener una longitud *L* manejable para facilitar su intercambio, y el algoritmo debe ser lo suficientemente eficiente como para generar el flujo de bits aleatorios a la velocidad que requiera la aplicación que lo utilice. Evidentemente, hay un compromiso entre la seguridad que se podrá alcanzar y la longitud del flujo de bits aleatorios obtenidos dada una longitud de clave.

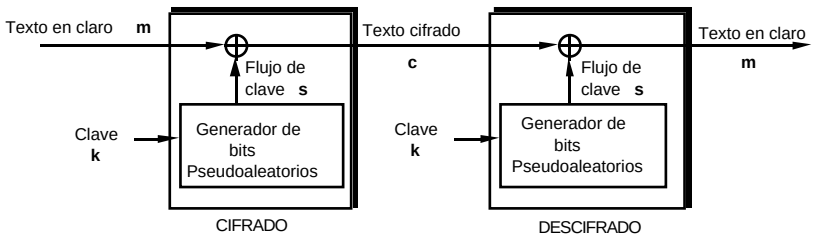


Fig. 7.2 Cifrador en flujo

Hay dos puntos cruciales a la hora de establecer los criterios de diseño de cifradores en flujo [ZEN91][RUE89]. El primero es la velocidad a la que se deben generar los bits (es decir, lo eficiente que debe ser el algoritmo de generación o, dicho de otra forma, cómo penaliza a la velocidad de la comunicación el hecho de poner el cifrador de flujo) y, en segundo lugar, lo seguro que se necesite que sea el sistema (cuya mejor medida es el tiempo que se tardaría en romperlo).

Un cifrador de flujo rompe el mensaje en claro m ($m \in M$) en un flujo sucesivo de caracteres m_1, m_2, m_3,..., y cifra cada carácter m_i mediante una transformación T variante en el tiempo bajo el control del símbolo s_i de la secuencia pseudoaleatoria en el instante i, con lo que se obtiene un nuevo carácter de la secuencia de texto cifrado $c_i = T(m_i, k_i)$. Tras cada carácter cifrado, el dispositivo cambia de estado interno de acuerdo con una cierta regla. En esta aplicación, a la secuencia pseudoaleatoria se

la denomina flujo de clave. El flujo de clave (o *keystream* en inglés) $s = s_1, s_2, s_3,...$ se obtiene a partir de la clave secreta k ($k \in K$). El mecanismo por el cual se expande la clave secreta k, que suele ser de corta longitud, en un gran flujo de bits pseudoaleatorio s, se le llama generador pseudoaleatorio. Los cifradores de flujo se usan ampliamente para obtener la confidencialidad de la información.

Se puede concluir que la transformación T consiste en combinar los caracteres que constituyen el mensaje a transmitir con los caracteres de salida de un generador de bits pseudoaleatorios y éste consistirá en una familia de funciones $\{G_L : L \in N\}$ que extenderá una clave k ($k \in K$) de longitud corta L a una secuencia s de longitud P mucho mayor:

$$G_L : \{0,1\}^L \rightarrow \{0,1\}^P \qquad\qquad s^P = G_L(k^L)$$

7.1.3 Modos de operación de un cifrador en flujo

La teoría de autómatas y, en especial, la de máquinas de estado finito es muy útil para describir los sistemas de cifrado en flujo y sus diferentes modos de operación [RUE86]. Los alfabetos empleados para el texto en claro y para el texto cifrado se determinan normalmente en función de la aplicación que se considere, aunque en la mayoría de los casos se usa el alfabeto binario para ambos, ya que es el utilizado en comunicaciones digitales. Los cifradores en flujo se dividen normalmente en dos tipos: los síncronos y los autosincronizantes. A continuación se decriben ambos tipos.

A) Cifradores de flujo síncronos

Tal como se vio en el capítulo 3, un generador de secuencias pseudoaleatorias se puede dividir

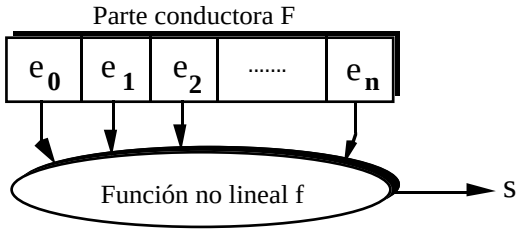


Fig. 7.3 Estructura de un generador de secuencias pseudoaleatorias

en una parte conductora F que es la que cambia de estado según una determinada regla, y una función de estado no lineal f que hace que la secuencia producida por la parte conductora sea más impredictible, tal como se puede ver en la figura 7.3. Por tanto, el proceso de generación del flujo de bits de clave se puede representar como $G = (F, f)$. Puesto en notación abreviada, donde se establece la relación funcional entre la clave k y el flujo de claves s^P:

$$G : k \rightarrow s^P \qquad\qquad s^P = G(k)$$

donde k representa una clave escogida de un espacio de claves K, y E denota el espacio de estados internos del generador (sus distintos estados internos son los elementos: e_0, e_1, e_2,..., donde e_0 es el estado inicial).

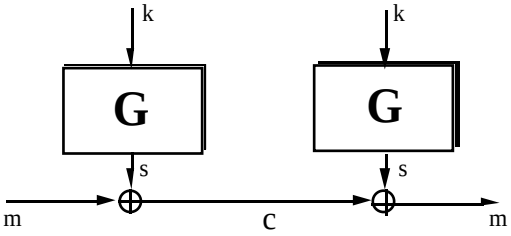


Fig. 7.4 Cifrador en flujo síncrono

En un cifrador de flujo síncrono, el flujo de claves (*keystream*) se genera independientemente del flujo de caracteres del mensaje (y de la secuencia de texto cifrado), tal como se muestra en la figura 7.4. La generación de la secuencia de claves está gobernada por dos reglas:

$$e_{i+1} = F(k, e_i)$$
$$s_i = f(k, e_i)$$

Los cifradores de flujo síncronos se pueden dividir a su vez de acuerdo con el modo que operan:

A1) Modo contador

$$e_{i+1} = F(e_i)$$
$$s_i = f(k, e_i)$$

En este caso el siguiente estado de la parte conductora no depende de la clave pero está garantizado que pasará a través de todos (o la mayoría) de los estados del espacio de estados. Ejemplos de tales funciones conductoras son los contadores y los LFSR de longitud máxima. La fortaleza criptográfica reside, sin embargo, en la función de estado no lineal de salida f.

A2) Modo de realimentación de salida o de realimentación interna

En este modo se cumple que:

$$e_{i+1} = F(k, e_i)$$
$$s_i = f(k, e_i)$$

Ahora la función de estado no lineal f no depende de la clave. Frecuentemente este método consiste simplemente en usar un bit del estado actual como predicado para la realimentación (por ejemplo, el bit menos significativo o de paridad).

A veces se usa una variante de este modo donde la clave *k* determina sólo el estado inicial:

$$e_0 = k$$
$$e_{i+1} = F(e_i)$$
$$s_i = f(e_i)$$

En los sistemas de flujo síncronos, si un carácter del texto cifrado se pierde, tanto el emisor como el receptor deben resincronizar sus generadores de claves. Si se pierde la sincronización, debe recuperarse inicializando todo el sistema, pues no se recupera por sí sola. Esto garantiza que ante cualquier ataque se desincronizará y no se podrá alterar el contenido del mensaje.

Por tanto, los cifradores en flujo síncronos son inmunes a ataques activos de inserción, eliminación y repetición de parte de la secuencia del texto cifrado, garantizando así la integridad de la información.

B) Cifradores de flujo autosincronizantes

A diferencia de los anteriores, en los que cada nuevo símbolo de salida del generador de secuencias pseudoaleatorias venía determinado sólo en función del estado interno y de la clave, en los cifradores autosincronizantes el nuevo estado (y, por tanto, la salida del generador) depende además de *N* símbolos previos del texto cifrado, tal como se ve en la figura 7.5. El modo más común de cifrador de flujo autosincronizante es el de realimentación de texto cifrado, que es el que muestra la figura 7.5, y que cumple que:

$$e_i = F(c_{i-1}, c_{i-2}, \ldots, c_{i-N})$$
$$s_i = F(k, e_i)$$

La fortaleza criptográfica reside en la función de estado no lineal f. Hay que notar que la entrada (los símbolos del texto cifrado) y la salida (el flujo de clave) de la función de estado no lineal f son conocidos por el criptoanalista en un ataque con texto en claro conocido.

Con este tipo de sistemas, si se altera o se pierde un bit (o carácter) del texto cifrado antes de que éste llegue al receptor, el error se propagará durante los *N* bits (o caracteres) descifrados siguientes, pero el cifrador se resincronizará una vez reciba de nuevo *N* bits (o caracteres) correctos del texto cifrado.

Una propiedad muy importante de este tipo de cifradores es que la secuencia de salida no es periódica debido a la dependencia que tiene el texto cifrado con los caracteres precedentes (y, por tanto, con el mensaje, el cual será con toda seguridad no periódico).

Las ventajas del método autosincronizante son: es capaz de recuperarse de los errores que se hayan producido en el canal debido a que el tamaño de su memoria interna es limitado, y protege al mensaje contra ataques basados en archivos de cifrado, por lo que estará protegido contra búsquedas.

Sus desventajas son la dificultad que presentan para ser analizados, incluso por el propio diseñador (ya que todas sus características dependen del mensaje que se transmite), y el hecho de que no son capaces de proteger contra posibles ataques activos de inserción, eliminación y repetición de textos.

Siempre será posible obtener un sistema autosincronizante a partir de otro síncrono, simplemente modificando el punto de realimentación del registro interno del sistema.

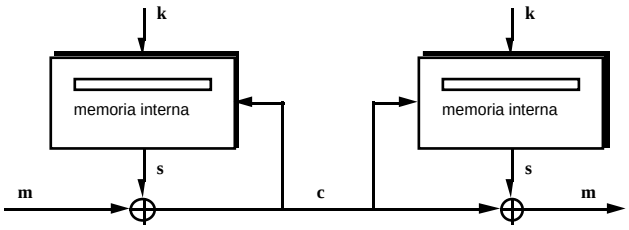


Fig. 7.5 Cifrador en flujo autosincronizante

7.1.4 Requerimientos de seguridad

Los cifradores de flujo presentan una gran utilidad para el cifrado de datos asíncronos y datos sin estructura de trama en un amplio rango de velocidades, que va desde datos a baja velocidad hasta velocidades elevadas del orden de incluso 150 Mbits/seg, o mayores.

La mayoría de las publicaciones en el tema coinciden en identificar tres consideraciones referentes a la seguridad que debe cumplir un cifrador en flujo [BEK82][ZEN91][SHA81]:

R1) El número de claves posibles debe ser suficientemente grande como para que el criptoanalista desista de probarlas todas en un ataque por fuerza bruta.

R2) Su periodo deberá ser al menos tan largo como el mensaje que debe cifrar, en analogía con el sistema de Vernan.

R3) El texto cifrado debe ser impredictible.

Se deberá intentar que las secuencias pseudoaleatorias producidas por el generador escogido para el cifrador en flujo garantice el cumplimiento de los tres requisitos anteriores. R1 es fácil de conseguir, ya que en la especificación del diseño se puede imponer el uso de longitudes de clave (L) suficientemente largas, e imponer que las 2^L combinaciones binarias que forman el espacio de claves sean posibles y equiprobables. El segundo requisito tampoco es difícil de lograr, ya que se puede hacer que las estructuras generen un periodo tan grande como queramos actuando sobre algunos de sus parámetros de diseño, tal como se vio en el capítulo 4, para todos los generadores estudiados. También se vio que una forma de hacer crecer el periodo de las secuencias obtenidas era con el uso de técnicas de crecimiento en cascada de generadores. En cualquier caso, aparecerán compromisos entre los requisitos R1 y R2 si se considera que debe haber una gestión de claves (tanto más cómoda cuanto menor sea el número de éstas) o si hay limitaciones tecnológicas o de volumen (en cuyo caso el crecimiento que hagamos de la estructura del generador no puede ser descuidada).

El requisito R3 es realmente la piedra angular en el diseño de los cifradores en flujo. Puesto que en la práctica los algoritmos que se utilizarán para generar las secuencias aleatorias serán deterministas y generarán el flujo de bits al ser inicializados con una clave secreta finita, puede ocurrir que no todas las posibles secuencias de bits que generen sean independientes. Esto implicaría que cada

criptograma no representase de igual forma un mensaje dado, con lo que la interceptación de este criptograma podría dar información acerca del mensaje. Si el criptoanalista realizase un ataque con texto en claro conocido podría, a partir de éste y de su correspondiente texto cifrado, obtener una porción del flujo de bits pseudoaleatorios y, si la secuencia no fuese impredictible, obtener a partir de ellos información sobre otras partes de este flujo de bits. Este tipo de ataques representan la principal debilidad de los cifradores en flujo.

Por tanto, un requerimiento fundamental es que el conocimiento de que el flujo de bits aleatorio haya sido generado mediante un algoritmo determinista conocido, no debe proporcionar ninguna utilidad para la extracción de la información del mensaje a partir de un criptograma interceptado.

Algunas de las propiedades en las que se pone mayor énfasis cuando se proponen diseños de generadores de secuencias pseudoaleatorias para sistemas de cifrado en flujo para evaluar el cumplimiento del requisito R3 son el periodo, la complejidad lineal y el crecimiento de ésta, la aleatoriedad o distribución estadística de los bits y las propiedades de correlación. Será de esperar que las secuencias empleadas por sistemas de cifrado en flujo tengan un buen comportamiento respecto a todas estas propiedades.

7.1.5 Principios de diseño de los cifradores en flujo

En el diseño de generadores de secuencias pseudoaleatorias para cifrado en flujo se deben considerar dos requisitos fundamentales [RUE89]. El primero es de tipo práctico, necesario en cualquier sistema y que determina si el diseño final es rápido y fácil de implementar.

El segundo debe asegurar que es difícil de romper (por ejemplo, mostrando un tiempo de criptoanálisis elevado). Este determina el tiempo necesario para que un criptoanalista sea capaz de romper el sistema.

Se pueden distinguir cuatro aproximaciones en el diseño y construcción de sistemas de cifrado en flujo [RUE86][RUE89][RUEPL] que difieren en las suposiciones que hacen sobre la capacidad y las oportunidades que tendría un posible criptoanalista (atacante) que pretendiese criptoanalizar (romper) el sistema, en la definición de lo que se considera un ataque criptoanalítico con éxito y en la noción de seguridad.

A) Cifradores de seguridad incondicional (o teórica)

Los cifradores de seguridad incondicional son aquellos en los que un oponente no puede obtener la distribución de probabilidad de los mensajes en claro en un ataque con sólo texto cifrado, incluso teniendo recursos de computación ilimitados. Esto significa que el texto en claro y el texto cifrado serán independientes.

Esta aproximación presenta los siguientes problemas: en primer lugar, la seguridad frente a un ataque con sólo texto cifrado no implica necesariamente seguridad frente a otro tipo de ataque, como por ejemplo los que se llevan a cabo cuando el texto en claro es conocido. En segundo lugar, para alcanzar la seguridad incondicional, la clave debe ser más larga que el mensaje, lo cual hace el criptosistema impracticable.

Finalmente, asumir oponentes con recursos de computación ilimitados es bastante restrictivo. En este caso, probablemente no existirían los criptosistemas de clave pública.

B) Cifradores computacionalmente seguros

Los cifradores computacionalmente seguros son aquellos en los que se cumple que un atacante, bajo la suposición de que disponga de un cantidad razonable de recursos, no es capaz de distinguir entre la secuencia producida por el generador y una secuencia obtenida por el experimento de tirar monedas. Desde un punto de vista práctico, el cifrado con un flujo binario pseudoaleatorio computacionalmente seguro es tan bueno como el cifrado realizado con una secuencia verdaderamente aleatoria. Esta indistinguibilidad implicará que el criptoanalista no pueda esperar desarrollar ningún ataque que pueda ser exitoso.

En esta aproximación, el generador de secuencias pseudoaleatorias se puede modelar mediante un parámetro de seguridad (generalmente la longitud de la clave), y se lleva a cabo un análisis de seguridad asintótico basado en la suposición de que todo algoritmo cuyo tiempo de ejecución pueda expresarse de forma polinómica en función de la clave es computacionalmete factible. Esta aproximación introduce los siguientes problemas: en primer lugar, puesto que es imposible probar que todos (o casi todos) los ejemplos de un problema requieren tiempo superpolinomial en el tamaño de la clave, se ha tenido que recurrir al recurso de usar *hipótesis de intratabilidad* para ser capaces de probar la seguridad computacional. De hecho, no hay publicado ningún generador que se pueda demostrar que es computacionalmente seguro sin tener que recurrir a estas hipótesis no demostradas. La cuestión está también en qué medida se prueba mediante un análisis asintótico, ya que todas las implementaciones de criptosistemas tienen por fuerza tamaños finitos. Debe haber ataques de tipo exponencial que sean realizables computacionalmente para los tamaños de los sistemas que pueden implementarse.

En los últimos años ha surgido un sistema teórico de complejidad que define como perfecto o criptográficamente seguro a un generador pseudoaleatorio si pasa todos los tests estadísticos que funcionen en tiempo polinomial. En otras palabras, la distribución de la secuencia de salida de un generador perfecto no debe poder distinguirse, al utilizar algoritmos de chequeo que funcionen en tiempo polinómico, de la distribución uniforme de secuencias de la misma longitud. A primera vista, parece un esfuerzo imposible demostrar que un generador pasa todos los tests estadísticos.

C) Cifradores prácticamente seguros

Los cifradores prácticamente seguros son aquellos que no se puede romper (en un tiempo útil) mediante cualquiera de los ataques criptoanalíticos comúnmente conocidos. Esta aproximación introduce también una serie de problemas. En primer lugar, puede que el diseñador de los criptosistemas no conozca todos los ataques de que dispone el posible atacante. Así pues, la experiencia del diseñador juega un papel importante en la seguridad de los criptosistemas. En segundo lugar, se pueden desarrollar métodos criptoanalíticos nuevos e imprevisibles que hagan obsoleto el criptosistema. Este comentario también se puede aplicar a la aproximación por complejidad. Algunos problemas que se consideran difíciles de resolver pueden resultar tratables, convirtiendo en no utilizable la correspondiente hipótesis de intratabilidad. Por tanto, la aproximación teórica al sistema define un cifrador de flujo como prácticamente seguro si resiste todos los ataques criptoanalíticos comúnmente conocidos, prescindiendo de sus características de tiempo de ejecución. Ello lleva a un primer objetivo consistente en desarrollar métodos y principios que tengan propiedades probables con respecto a ciertas medidas teóricas del sistema, como la complejidad lineal, el periodo, la inmunidad a la correlación, la transformada de Walsh, etc. Un segundo objetivo será la necesidad de desarrollar reglas de diseño para que esos ataques no sean practicables.

D) Cifradores de seguridad probable

Se dice que los cifradores son de seguridad probable si se puede demostrar que existe una cota inferior (en media) en los pasos que debe realizar para llevar a cabo cualquier ataque. La demostración de la cota inferior no debe recurrir a ninguna hipótesis no demostrada. Desafortunadamente, hasta el momento no se ha encontrado ningún cifrador de flujo práctico de seguridad probable. La seguridad probable se refiere a la de ataques que estén limitados por la cantidad de datos o por el tiempo de que se dispone. Así pues, se deberán diseñar cifradores de flujo con una cota inferior para los recursos que un atacante deba emplear o a la información que necesitará para romper el sistema.

Es posible que la secuencia producida por un generador de secuencias pseudoaleatorias presente una aleatoriedad local perfecta, por lo que no será posible realizar un ataque eficiente usando menos que esta información. Es interesante, pues, determinar cuán cerca estará un ataque que pueda ser exitoso de un ataque basado en la búsqueda exhaustiva.

Asociada a la suposición de la existencia de limitaciones en los recursos disponibles para atacar un cifrador en flujo que era probablemente seguro, está también la consideración de que será también seguro contra los tests más conocidos que considera la aproximación teórica al sistema.

Sin embargo, hay que tener precaución en esto, puesto que seguridad probable no quiere decir lo mismo que seguridad incondicional, ya que la seguridad probable se mide en función de los recursos que se espera pueda necesitar un atacante, como mínimo, para romper el sistema.

7.1.6 Criptoanálisis

El criptoanálisis consiste, en última medida, en la búsqueda de la clave del sistema de seguridad. El método más burdo que se podría ocurrir para romper un criptosistema podría ser probar todas las posibles combinaciones de claves, hasta que se diera con la correcta.

A este procedimiento se le llama ataque por fuerza bruta [HEL84] y, aunque presupone que se conoce el algoritmo que se está usando, sin duda no parece ser una forma útil para atacar el sistema.

El problema se puede trasladar al hecho de buscar soluciones que minimicen la cantidad total de trabajo necesario para encontrar, de forma exitosa, la clave del sistema. Si toda la clave secreta k de un sistema se puede obtener mediante una búsqueda exhaustiva concentrada en una cierta parte de ésta (subclave) k_1, significará que sólo $|k_1|$ de estos bits de clave serán los responsables de la seguridad criptográfica del sistema, y los restantes $|k| - |k_1|$ bits de la clave serán redundantes.

A la relación:

$$\rho = \frac{|k| - |k_1|}{|k|}$$

se la llama la *tasa de redundancia* en la información de la clave del sistema. Los sistemas que se pueden romper mediante ataques analíticos puros presentan una redundancia de información de la clave $\rho = 1$, aunque estos casos se dan raramente. El problema residirá en encontrar la redundancia $|k| - |k_1|$.

La mayoría de los generadores de secuencias pseudoaleatorias descritos en el capítulo 4 han sido criptoanalizados con éxito (rotos) utilizando algún método basado en encontrar la redundancia de la clave y reduciendo así el tiempo que se emplearía en romperlo utilizando la búsqueda exhaustiva

[ZEN91][BOY89]. Esto no quita para que muchos de ellos se estén usando de forma efectiva en sistemas actuales en uso, ya que el hecho de usar un sistema de gestión de claves eficiente que cambie éstas lo suficientemente rápido es una contramedida eficiente para evitar este tipo de ataques. Lo ideal, sin embargo, sería elegir un algoritmo que fuese, en principio, resistente a cualquier tipo de ataque. Sin embargo, el problema de los cifradores de flujo es que, aunque podamos obtener un generador de secuencias pseudoaleatorias que sea resistente contra los ataques criptoanalíticos conocidos, no podemos garantizar que no pueda encontrase algún método matemático que sea capaz de encontrar alguna debilidad en la aleatoriedad de la secuencia que produce y mejorar las condiciones del ataque por fuerza bruta.

A continuación se describen (aunque de forma breve y descriptiva debido a la complejidad que un análisis más profundo implicaría) algunos de los métodos criptoanalíticos más ampliamente utilizados para criptoanalizar sistemas de cifrado en flujo.

A) El test de consistencia lineal (LCT)

Este ataque criptoanalítco, propuesto por Zeng, Yang y Rao [ZEN89], se basa en un método algebraico que aprovecha la linealidad existente en las secuencias producidas por los generadores de secuencias pseudoaleatorias. El método resulta muy efectivo para romper generadores tales como el generador multivelocidad de Massey-Ruepple y el generador de Jennings [ZEN91][ZEN89].

Si bien ya conocemos el método de búsqueda exhaustiva para encontrar una clave k, el LCT obtiene la ventaja mediante el análisis de una porción más pequeña de esta clave, probándola de forma exhaustiva disponiendo de un criterio que permita darla, o no, por válida.

Para ello, se realiza una estimación precisa de la probabilidad de consistencia de un sistema algebraico de ecuaciones lineales $Ax = b$, donde A es una matriz aleatoria obtenida a partir de una pequeña parte k_1 de la clave completa k y de tamaño $m \times n$ ($m>n$), y b es un vector fijo diferente de cero. Para este sistema de ecuaciones se cumplirá que:

1) La matriz A de coeficientes dependerá tan sólo de la subclave k_1.
2) El vector b se obtendrá a partir de un cierto segmento de la secuencia de salida.

Si la subclave k_1 no se elige correctamente, el sistema $Ax = b$ será inconsistente con una probabilidad cercana a 1. Sin embargo, si resulta ser consistente, se puede deducir de su solución la parte restante de la clave k, es decir, k-k_1. Además, este hecho implicará que $|k| - |k_1|$ bits de la parte restante de la clave k no contribuirán sustancialmente a la seguridad del sistema global, y se habrá descubierto una cierta redundancia en la clave. Por tanto, el metodo consistirá en aplicar una búsqueda exhaustiva para determinar la subclave k_1, usando como criterio de identificación la consistencia del sistema de ecuaciones lineal correspondiente y si esto resulta exitoso, la solución del sistema permitirá determinar el resto de la clave k.

B) El ataque por correlación de Siegenthaler

En muchos generadores estudiados en este libro, la secuencia pseudoaleatoria $z(t)$ se obtenía combinando las secuencias producidas por otros subgeneradores $x_i(t)$ ($1 \leq i \leq k$). Ya se vio que existirá una dependencia estadística entre $z(t)$ y cada $x_i(t)$ [GEF73], que será más o menos fuerte según el buen o mal criterio que se haya seguido a la hora de elegir la función combinadora [MEI92].

En algunos generadores, como los de Geffe [GEF73], Bruer [BRU82] o Pless [PLE77] [RUB79], que se vieron en el capítulo 4, esta dependencia estadística entre la secuencia de salida y las subsecuencias que se usan (que en los tres casos son la salidas de los LFSRs) es bastante acusada y se puede aprovechar para establecer un tipo de ataques conocidos comúnmente como de divide y vencerás [SIE85] para poder determinar la clave. Su nombre viene de la forma en que se realizan, ya que se basan en analizar separadamente las diferentes subsecuencias $x_i(t)$. Además, el hecho de dividir el problema en otros de menor envergadura, permite abordar cada uno de ellos empleando la búsqueda exhaustiva.

La relevancia criptográfica de este hecho fue presentada por primera vez por Blaser y Heinzman [BLA82], y estudiada porteriormente por Siegenthaler [SIEGE], quien establece que la secuencia de salida $z(t)$ se puede ver como una perturbación sobre cada una de las subsecuencias $x_i(t)$, y que esta perturbación vendrá definida por una fuente de ruido asimétrica y sin memoria con probabilidad p (ésta será además la probabilidad correspondiente de correlación). Si la probabilidad p difiere del valor 1/2, será posible para un atacante descubrir subsecuencias $x_i(t)$ (desconocidas para él) calculando la correlación de cada una de sus posibilidades con la secuencia de salida $z(t)$, y aceptando la candidata que dé un valor en esta correlación por encima de un umbral [STA89].

Hay que observar que para realizar este tipo de ataque, el criptoanalista debe disponer de una cierta porción de la secuencia $z(t)$ y si a partir de dicha porción de la secuencia el ataque resulta exitoso, podrá determinar las subsecuencias y a partir de ellas, puesto que ya dispondrá de la información completa del generador y de las claves que se han empleado, será capaz de determinar toda la secuencia $z(t)$.

Si el atacante sabe que las subsecuencias empleadas las producen LFSRs, y además conoce su estructura (por ejemplo, en el caso de que el algoritmo de cifrado sea de dominio público), sólo le faltará obtener las claves (los estados iniciales de los LFSRs), y para ello deberá tan sólo probar de forma exhaustiva todos estos posibles estados iniciales para cada LFSR. Si el LFSR tiene n celdas, ello implicará probar 2^n-1 posibilidades para cada uno de los LFSRs (aunque en media, tan sólo deberá probar la mitad antes de dar con la correcta). En la práctica, este ataque sólo tendrá validez para LFSRs de hasta $n = 60$ celdas.

Debido al enorme éxito obtenido mediante este tipo de ataques sobre generadores basado en funciones combinadoras, Siegenthaler y Ruepple proponen [MAT88][CAM78] el uso de funciones que tengan máxima inmunidad a la correlación.

Este concepto, que ya se vio en el capítulo 3, establece que si se combinan N LFSRs mediante una función combinadora que cumpla esta característica, no debe existir correlación entre la secuencia de salida y las secuencias producidas por cualquier subconjunto de menos de N LFSRs. Las funciones combinadoras que cumplan esto, producirán secuencias que no se podrán atacar por este tipo de técnicas del tipo divide y vencerás.

Sin embargo, recientemente Meier y Staffelbach [MEI92][MEI89] desarrollaron dos algoritmos que permiten descubrir esta correlación entre la salida de cada LFSR y la secuencia de salida del generador de una forma mucho más efectiva que la de la búsqueda exhaustiva en el caso de que los LFSR contengan funciones de realimentación a los que contribuyan pocas celdas, pudiendo llegar a atacar LFSRs con $n = 1.000$ e incluso más celdas. Otro algoritmo eficiente para este tipo de ataques es el que proponen Chepyzhov y Smeets [CHEPY].

La aparición de este tipo de ataques nos lleva a considerar la máxima inmunidad a la correlación como una condición no suficiente para evitarlos, por lo que hay que ampliar los requisitos de diseño para este tipo de generadores:

1) Debe existir inmunidad a la correlación de máximo orden respecto a las salidas de los LFSR.

2) No debe existir correlación entre la secuencia de salida y cualquier LFSR que tenga menos de 10 celdas contribuyendo a su realimentación .

3) No debe existir correlación entre la secuencia de salida y cualquier LFSR con menos de 100 celdas.

El ataque propuesto por Siegenthaler [SIEGE] es, pues, efectivo sobre generadores basados en funciones combinadoras con inmunidad a la correlación de orden cero (cuando la salida está correlada con al menos una entrada) y se basa en medir la distancia de Hamming entre la secuencia de salida y un trozo del texto cifrado que el atacante haya podido conseguir.

Tal como se vio en el capítulo 4, existen otro tipo de generadores que se basan en emplear técnicas de reloj para generar las secuencias pseudoaleatorias. El hecho de usar relojes con pautas irregulares reduce considerablemente el peligro frente a ataques basados en el estudio de la correlación. Sin embargo, hay que tener cuidado al emplear LFSRs atacados por relojes no uniformes y combinados mediante funciones sin memoria, puesto que existen ataques basados en la distancia restringida de Levenshtein [GOL91] que permiten también realizar ataques de correlación sobre ellos. Si suponemos que las operaciones que se pueden emplear sobre una secuencia para transformarla en otra son la sustitución, el borrado y la inserción de elementos, se puede definir la distancia de Levenshtein entre dos secuencias como el mínimo número de estas operaciones necesarias para transformar una secuencia en la otra. A partir de esta definición, se puede establecer la distancia restringida de Levenshtein entre dos secuencias u (de longitud M) y v (de longitud N), con $N \leq M$, como la mínima suma de distancias asociadas con las operaciones sustitución, borrado e inserción necesarias para obtener una secuencia a partir de la otra, bajo la restricción de que el máximo número de borrados posibles consecutivos sea E. Esta definición lleva inherente también el número máximo de inserciones posibles (M-N) y de sustituciones (N). Además, se deberá cumplir la restricción $N \leq M \leq (E+1)N+$E.

Basándose en esta distancia, Golic y Mihaljevic establecen en [GOLIC] un algoritmo recursivo para realizar un ataque por correlación sobre LFSRs con relojes no uniformes combinados con funciones sin memoria con inmunidad a la correlación de orden 0.

C) El método del síndrome lineal

Puesto que los métodos anteriores utilizan la búsqueda exhaustiva para determinar la clave de un criptosistema, son métodos más apropiados para descubrir la redundancia en la clave de los diversos generadores pseudoaleatorios que algoritmos prácticos para recuperar el texto en claro en un sistema de cifrado en flujo.

Sin embargo, si la dependencia estadística entre la secuencia pseudoaleatoria empleada y la secuencia de texto en claro a cifrar es suficientemente grande, se puede usar de forma muy efectiva un ataque analítico basado en el método del síndrome lineal propuesto por Zeng y Huang [ZEN88]. A continuación se describen las bases de este método.

Supongamos que la fuente de texto en claro, que llamamos $m=\{m(t)\}$, está estadísticamente desbalanceada, de forma que contiene mayor número de ceros que de unos. La probabilidad de coincidencia entre la secuencia pseudoaleatoria (que llamaremos $s(t)$) y la secuencia del texto cifrado $c(t)$ (que se obtendrá mediante la operación $c(t) = m(t) \oplus s(t)$ en un cifrador en flujo como ya conocemos) vendrá determinada por:

$$p = prob(s(t) = c(t)) = \frac{1}{2} + \varepsilon \qquad \text{con } \varepsilon > 0$$

Al número ε se le denomina la *ventaja en la coincidencia* entre las secuencias $s(t)$ y $c(t)$. El objetivo a alcanzar radica en obtener la secuencia $s(t)$ a partir de la secuencia $c(t)$ incrementando esta ventaja.

Para ello, se debe asumir que el polinomio primitivo $f(x)$ del LFSR del generador de secuencias pseudoaleatorias es conocido, pero no lo es su estado actual.

Entonces, se puede considerar un conjunto de múltiples trinomiales (es decir, de tres términos) de $f(x)$:

$$g(x) = 1 + x^h + x^k \qquad k > h > 0$$

y a partir de cada uno de estos trinomios, se pueden obtener las siguientes tres señales:

$$\sigma_1 = c(t) \oplus c(t+l) \oplus c(t+k)$$
$$\sigma_2 = c(t-l) \oplus c(t) \oplus c(t+k-l)$$
$$\sigma_3 = c(t-k) \oplus c(t-k+l) \oplus c(t)$$

que se denominan síndromes. Zeng y Huang [ZEN88] demuestran que la ventaja en la coincidencia entre cada $\sigma_i(t)$ y $s(t)$ es $2\varepsilon^2$ que, aunque es más pequeña que la ventaja de partida ε, es más conveniente, ya que se puede intentar amplificarla si se considera el efecto global que se tendría al tomar un conjunto tanto más y más grande de síndromes. El descubrimiento y amplificación de ventajas es un principio de gran importancia en el criptoanálisis.

Se puede concretar más este hecho si se considera un conjunto de $2m+1$ síndromes y se revisa la secuencia $c(t)$ de acuerdo con la siguiente regla de decisión mayoritaria:

$$c'(t) = \bar{c}(t) \quad si \quad al \quad menos \quad m+1 \quad síndromes \quad son \quad 1$$
$$c'(t) = c(t) \quad en \quad otro \quad caso$$

En [ZEN88] se demuestra que la ventaja en la coincidencia ε_m entre $s(t)$ y $c`(t)$ tenderá a 1/2 cuando m crezca indefinidamente. Esto implica que seremos capaces de recuperar la secuencia $s(t)$ (y, por tanto, la secuencia $m(t)$) a partir de la secuencia $c(t)$ tan sólo incrementando el número de síndromes que se usen.

Se podría hacer, sin embargo, un acercamiento más práctico [ZEN90] mediante el uso de revisiones que interactúen entre sí, utilizando ahora un número fijo adecuado de síndromes calculados a partir de fórmulas fijas. En particular, cuando ε es suficientemente grande (por ejemplo ε=0,25), será fácil recuperar la secuencia $s(t)$ a partir de un segmento de la secuencia $c(t)$ con una longitud

linealmente proporcional con el número de celdas L del LFSR que se quiere atacar. Además, el esfuerzo computacional que se deberá realizar también será linealmente proporcional con L.

Con este método se pueden romper con facilidad sistemas de cifrado en flujo que usen generadores de secuencias pseudoaleatorias como el generador de Geffe o el generador de marcha y espera de Beth Piper [ZEN91].

Ya se han visto los tres tipos de ataques más comunes para obtener la redundancia en la clave que pueda presentar un sistema de cifrado en flujo y reducir así el esfuerzo que requerirá la búsqueda exhaustiva de la clave. Sin embargo, el hecho de que un generador no sea roto por ellos, no significa que no se pueda determinar otro tipo de metodología que permita romperlo. Sin embargo el hecho de que resista estos ataques nos dará más y más información sobre la ausencia de esta redundancia y, por tanto, de su fortaleza criptográfica. Algunos generadores que se han mostrado resistentes a los tres tipos de ataques que se han descrito son la cascada de Gollmann y el generador de marcha y espera con control bilateral (ambos se describieron en el capítulo 4) [ZEN91].

7.1.7 Algunas aplicaciones del cifrado en flujo

Si bien en los apartado anteriores se ha descrito, aunque muy brevemente debido a la enorme amplitud del tema, la teoría básica del cifrado en flujo, puede ser interesante ver algunos sistemas de comunicaciones que usan este tipo de técnicas para proteger la información, y ver en qué forma usan los generadores de secuencias pseudoaleatorias para conseguirlo. Para ello se van a describir tres sistemas. En primer lugar se verá el sistema de acceso condicional del estándar MAC/paquetes para cifrar señales de televisión, bien sea a través de enlaces satélite o cable. En segundo lugar se verá una propuesta de generador de secuencias pseudoaleatorias para el sistema de cifrado en flujo de la futura Red de Banda Ancha Española ATM, dentro del proyecto nacional PLANBA (Plan Nacional de Banda Ancha). También se describirá el sistema de cifrado en flujo utilizado para conseguir confidencialidad en los sistemas de telefonía móvil GSM.

A) El sistema de acceso condicional Eurocrypt

Una aplicación que emplea el cifrado en flujo se encuentra en la difusión de señales tanto de imagen, como de voz y de datos a través de enlaces vía satélite y cable. La posibilidad de enviar servicios de forma encriptada permite hacerlos accesibles sólo a aquellos usuarios que cumplan ciertos requisitos, como por ejemplo haber pagado por ellos. Por tanto, a estos sistemas se les suele llamar de acceso condicional y deben realizar una serie de transformaciones sobre las señales en claro, de forma que las haga ininteligibles. Estas transformaciones estarán gobernadas por un parámetro llamado clave, de forma que las señales, una vez cifradas, sean inaccesibles para todos aquellos que no dispongan de esta clave. Un sistema de estas características es el sistema de acceso condicional Eurocrypt, asociado al estándar de transmisión de señales de televisión MAC/paquetes [EBU86][ALA86].

Este sistema de acceso condicional se puede dividir en dos partes por razones de conveniencia: el sistema de gestión de claves, que se encarga de distribuir las claves necesarias para cifrar los programas, y el sistema de cifrado, encargado de procesar las señales de los programas para hacerlas ininteligibles a los usuarios no autorizados, empleando para ello generadores de secuencias pseudoaleatorias de elevado periodo. Las celdas de los LFSRs que emplean estos generadores de secuencias pseudoaleatorias se cargarán inicialmente con los bits de lo que se llaman las *palabras de*

control (CW) y tienen 60 bits de longitud. Se trata, pues, de un sistema de cifrado en flujo y, por tanto, los procesos de cifrado en el emisor y de descifrado en el receptor deben estar sincronizados. Las señales cifradas, el contador de tramas (que se describirá a continuación) y las señales de sincronización se transmiten al receptor a través del mismo canal de transmisión.

El sistema de gestión de claves es el encargado de transmitir las claves entre el centro emisor y los abonados que las hayan adquirido. Para dar mayor seguridad al sistema, las palabras de control, que sirven de estado inicial de los generadores del cifrador y del descifrador, se cambian cada 256 tramas de televisión (lo que ocurre cada 10,24 segundos), pero se transmiten cada 0,5 segundos aproximadamente para permitir un rápido enganche al descifrado a los receptores que sintonicen con el canal. Para ello, se utiliza un contador de tramas cíclico de 8 bits (FCNT) que se transmite en la línea de televisión 625. Tal como muestra la figura 7.6, las palabras de inicialización de los generadores pseudoaleatorios se obtienen al combinar las palabras de control (CW) con el contador de tramas de 8 bits. Veamos cómo se produce esta combinación. Las funciones lógicas combinacionales que sirven para combinar las palabras de control (CW_1 y CW_2) con los estados del contador cíclico de 8 bits (FCNT) son las mismas tanto para el sistema de descifrado de imagen como para el de voz y datos. El resultado de esta combinación será la palabra de inicialización IW de 60 bits que se utilizará como inicialización a los LFSR. La combinación se realiza de la siguiente forma:

Sean $F_0, F_1,..., F_7$ los 8 bits del contador cíclico FCNT. Si se dividen los 60 bits de la palabra de inicialización en bloques de 8 bits: ($IW_0 = I_0I_1...I_7$, $IW_1 = I_8I_9...I_{15}$,..., $IW_6 = I_{48}I_{49}...I_{55}$, y $IW_7 = I_{56}I_{57}I_{58}I_{59}$) y lo mismo para la palabra de control ($CW_0 = C_0C_1...C_7$, $CW_1 = C_8C_9...C_{15}$,..., $CW_6 = C_{48}C_{49}...C_{55}$ y $CW_7 = C_{56}C_{57}C_{58}C_{59}$), entonces, estas palabras IW se obtienen de la siguiente forma:

$$IW_0 = CW_0 \oplus FCNT \qquad IW_1 = CW_1 \oplus \overline{FCNT} \qquad IW_2 = CW_2 \oplus FCNT$$
$$IW_3 = CW_3 \oplus \overline{FCNT} \qquad IW_4 = CW_4 \oplus FCNT \qquad IW_5 = CW_5 \oplus \overline{FCNT}$$
$$IW_6 = CW_6 \oplus FCNT \qquad IW_7 = CW_7 \oplus \overline{FCNT}$$

donde $\overline{FCNT}$ es el complemento de FCNT ($FCNT \oplus \overline{FCNT} = 11111111$).

En la transformación de CW_7 a IW_7, solamente se usan los cuatro bits menos significativos de $\overline{FCNT}$.

El sistema de cifrado utiliza generadores pseudoaleatorios de Jennings, que se estudiaron en el capítulo 4, y están constituídos por la combinación de registros de desplazamiento con realimentación lineal (LFSR) mediante un multiplexor.

El hecho de añadir el contador FCNT da mayor impredictibilidad a la secuencia de salida de los generadores pseudoaleatorios de Jennings. Un análisis de este generador se puede ver en el capítulo 9, donde se muestra que presenta ciertas deficiencias respecto a lo que se espera de las secuencias pseudoaleatorias utilizables en aplicaciones criptográficas. Sin embargo, el hecho de añadir el contador cíclico soluciona gran parte de estos problemas que presenta el generador de Jennings por sí solo.

La sincronización de los generadores pseudoaleatorios con el contador de tramas de 8 bits se realiza de la siguiente manera: consideremos dos tramas de televisión consecutivas F_n y F_{n+1}. Durante la línea 625 al final de la trama F_n, se recibe un nuevo valor del contador de trama (FCNT) y este valor se combinará con CW_1 y CW_2. Denotamos como CW_1 la palabra de control que se utilizará para el descifrador de la señal de video, y CW_2 la que se utilizará para descifrar el audio y los datos, aunque en realidad son la misma. El resultado de esta combinación serán los nuevos valores de IW_1 e IW_2, los

cuales se cargan en los generadores pseudoaleatorios de Jennings J_1 y J_2 respectivamente. Los primeros 61 bits generados por el generador J_1 los usará el generador J_3 como inicialización, y producirá entonces la secuencia que servirá para descifrar el primer paquete recibido durante la trama F_{n+1}. De igual forma los primeros 16 bits que producirá el generador J_2 después de ser inicializado con IW_2 se usarán para descifrar la primera línea de la trama F_{n+1}. Cada 256 tramas de televisión el sistema de generadores recibirá una nueva palabra de control que pasará a ser la válida cuando FCNT=0.

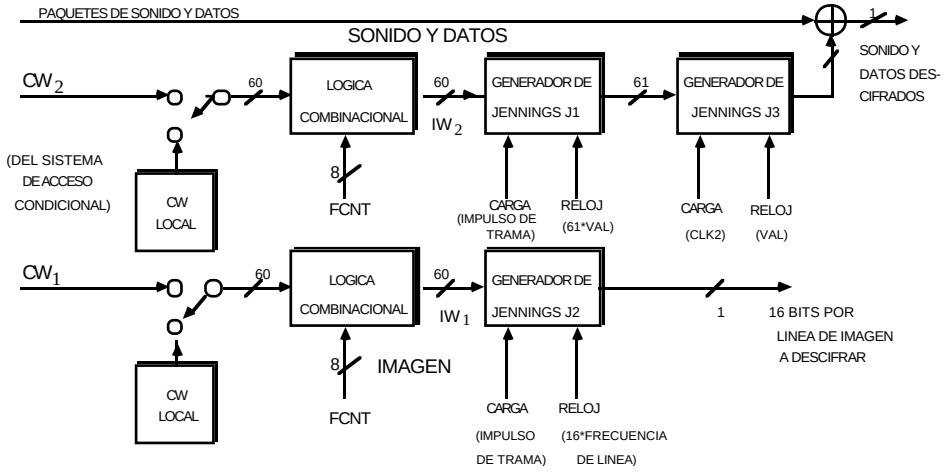


Fig. 7.6 Descifrador del sistema MAC/paquetes

El cifrado de la señal de imagen, que en el sistema MAC se transmite en forma analógica, se hace cortando su espectro en un punto y permutando las dos zonas espectrales resultantes. Los 16 bits de la secuencia pseudoaleatoria de salida del generador de Jennings J_2 determinan el punto de corte en el espectro de diferencia de color en el caso de corte simple o el punto de corte, tanto en el espectro de luminancia como en el de crominancia si se usa doble corte. Para el sonido y los datos, que se transmiten en forma digital, el encriptado se realiza combinando la secuencia producida por el generador formado por la cascada de los generadores de Jennings J_1 y J_3, mediante la suma módulo 2, con la información digital de audio y datos. A continuación se describirá la estructura de estos generadores pseudoaleatorios.

La figura 7.7 muestra la estructura del generador de Jennings J_1. En este generador, los polinomios de realimentación de los LFSR son:

$$SR_1 = 1+y+y^2+y^3+y^7+y^{14}+y^{19}+y^{25}+y^{31}$$
$$SR_2 = 1+x^2+x^3+x^4+x^8+x^{11}+x^{16}+x^{20}+x^{29}$$

Las palabra IW de 60 bits que sirve como inicialización para los LFSR SR_1 y SR_2 se define como sigue:

SR1: I_i se carga en la celda $y^{(31-i)}$ para i variando entre 0 y 30.
SR2: I_i se carga en la celda $x^{(60-i)}$ para i variando entre 31 y 59.

Las salidas de los registros atacan a las líneas de selección (denotadas por $A_0...A_4$) y a las de entrada (denotadas por $B_0...B_{31}$) del multiplexor. Las conexiones se realizan de la siguiente manera:

x_1 se conecta a A_0
x_2 se conecta a A_1

en general, para las x`s:

x_i se conecta a $B_{(i-3)}$ para i entre 3 y 10
y_1 se conecta a A_2
y_2 se conecta a A_3
y_3 se conecta a A_4

en general, para las y`s:

y_n se conecta a $B_{(n+4)}$ para n entre 4 y 27

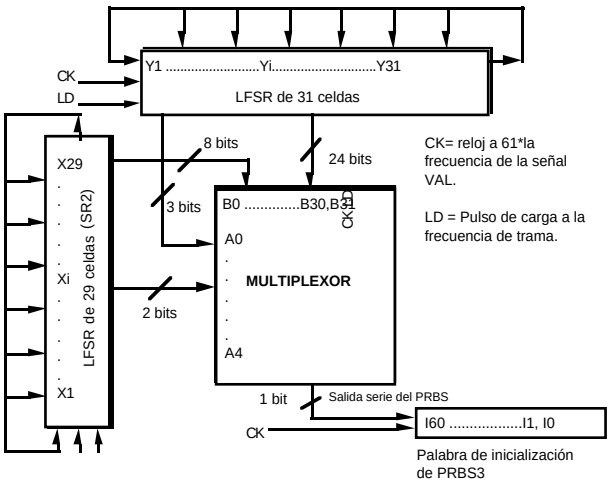


Fig 7.7 Generador de Jennings J_1

En la figura 7.8 se puede ver la estructura del generador de Jennings J_3. El polinomio de realimentación del LFSR es:

$$SR_5= 1+x^2+x^3+x^7+x^8+x^9+x^{10}+x^{12}+x^{15}+x^{19}+x^{20}+x^{22}+x^{24}+x^{25}+x^{28}+x^{30}+x^{33}+x^{34}+x^{37}+x^{40}+x^{43}+x^{44}+x^{46}+x^{54}+x^{56}+x^{60}+x^{61}$$

La palabra IW de 61 bits que sirve como inicialización para el LFSR SR_5 se define como $I_{60}...I_0$, o bien I_i, donde i toma valores entre 0 y 60. Así pues, la carga se produce como sigue: SR_5: I_i se carga en la celda $x^{(60-i)}$ para i variando entre 0 y 60.

Las salidas de los registros atacan a las líneas de selección (denotadas por $A_0...A_4$) y a las de entrada (denotadas por $B_0...B_{31}$) del multiplexor y las conexiones se realizan de la siguiente manera:

$x^{5(i+1)}$ se carga en A_i para i entre 0 y 4
$x^{(30+i)}$ se carga en B_i para i entre 0 y 31

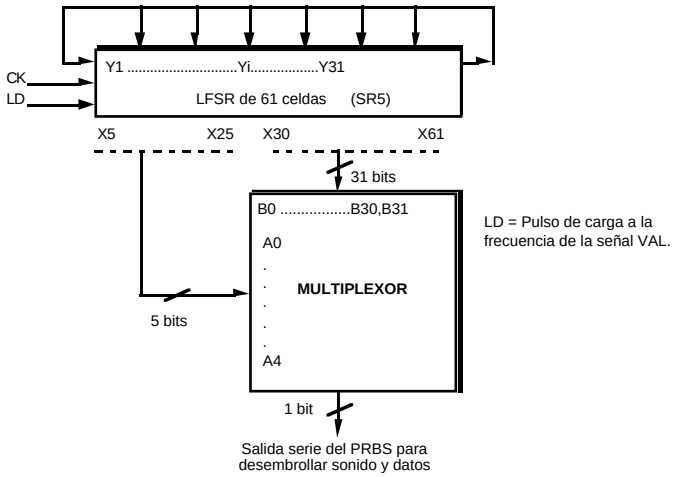


Fig. 7.8 Generador de Jennings J_3

Finalmente, la figura 7.9 muestra la estructura del generador de Jennings J_2 empleado para realizar el cifrado de la señal de imagen.

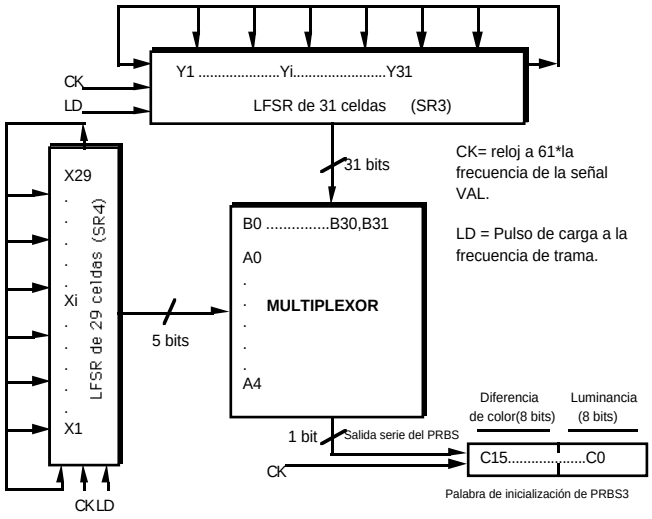


Fig. 7.9 Generador de Jennings J_2

Los polinomios de realimentación de los LFSR de este generador son:

$$SR_3 = 1+y+y^2+y^3+y^5+y^6+y^7+y^9+y^{10}+y^{11}+y^{15}+y^{19}+y^{23}+y^{27}+y^{31}$$
$$SR_4 = 1+x^2+x^3+x^4+x^5+x^7+x^{11}+x^{13}+x^{14}+x^{20}+x^{29}$$

Las palabras IW de 60 bits que sirven como inicialización para los LFSR. SR_3 y SR_4 se definen como:

SR_3: I_i se carga en la celda $y^{(31-i)}$ para i variando entre 0 y 30.
SR_4: I_i se carga en la celda $x^{(60-i)}$ para i variando entre 31 y 59.

Las salidas de los registros atacan a las líneas de selección (denotadas por $A_0...A_4$) y a las de entrada (denotadas por $B_0...B_{31}$) del multiplexor y las conexiones se realizan de la siguiente manera:

x_i se conecta a $A_{(i-1)}$ para i entre 1 y 15
y_i se conecta a $B_{(i+1)}$ para i entre 1 y 31

Mediante estos generadores de Jennings, con las modificaciones adicionales respecto a la estructura básica mostrada en el capítulo 4 consistente en usar cascadas y añadir un contador cíclico básico, se obtiene un sistema final cuyas propiedades criptográficas son bastante satisfactorias: buenas propiedades estadísticas, gran periodo, elevada complejidad lineal y ausencia de picos de correlación fuera de fase. Sin embargo, el generador de Jennings puede romperse mediante el método de consistencia lineal [ZEN89] visto anteriormente. Aplicando este método, bastan $N = m + 2^h$ bits de la secuencia de salida del generador, a los que se deben aplicar 2^{m+h} tests de consistencia para criptoanalizarlo (donde m y h son los parámetros del generador descritos en el capítulo 4). Es interesante ver cómo se protege el sistema frente a un ataque basado en este método y la respuesta está en el sistema de gestión de claves, que es el encargado de cambiar las claves que sirven de inicialización para los generadores vistos hasta ahora. Este cambio es lo suficientemente rápido como para no dar tiempo a realizar este tipo de ataque [CRU93]. Vamos a verlo estudiando los generadores de descifrado para la señal de imagen y para la voz y datos por separado [ZEN89].

A1) El sistema de cifrado de la señal de imagen

La estructura del circuito cifrador de la señal de imagen está constituída por un generador de Jennings con dos LFSR con $m = 31$ y $n = 29$ (generador J_2). La palabra de inicialización es de 60 bits y, antes de cargarse en las celdas, se combina con los estados de un contador cíclico de 8 bits (256 estados). Los bits de salida de este generador, se toman de 16 en 16 para descifrar cada línea de imagen. Vamos a hacer algunos cálculos:

El periodo máximo que podemos obtener con esta secuencia es: $P = 256\,x\,(2^{29}\text{-}1)\,x\,(2^{31}\text{-}1) = 2{,}951479\ 10^{20}$. Puesto que se necesitan 16 bits de la secuencia pseudoaleatoria para descifrar cada línea de imagen, la tasa de producción de bits v del sistema debe ser:

v = 625 líneas/imagen x 25 imágenes/seg x 16 bits de desembrollado/línea = 250.000 bits/seg

A esta velocidad, el generador tardaría 37.436.314 años en producir un periodo de la secuencia. El sistema de gestión de claves, hace que la palabra de control cambie cada 256 tramas de TV (10,24 seg) y, por tanto, en este tiempo se debe combinar con los 256 estados del contador cíclio. Esto implica que los estados iniciales del generador se inicializan cada trama de TV, es decir, cada 40 mseg. Para realizar un criptoanálisis a este generador, se necesitan $N > m + n2^h = 29 + 31 \times 2^5 = 1.021$ bits,

sobre los que hay que realizar $2^{m+h} = 2^{34}$ tests de consistencia, pero en un tiempo inferior a 40 mseg, lo cual actualmente es muy difícil.

A2) El sistema de cifrado de las señales de audio y datos

Este generador consiste en una cascada de dos generadores de Jennings que tienen las siguientes dimensiones: $m = 31$, $n = 29$ y $p = 61$. La palabra de inicialización es de 60 bits que, antes de cargarse en las celdas de los LFSR del primer generador (J_1), se combina con los estados del contador cíclico de 8 bits. Una vez cargados, el generador produce 61 bits que se cargan en las celdas del LFSR del segundo generador (J_3), y cuya secuencia de salida sirve para descifrar los paquetes digitales de sonido y datos (mediante la suma módulo 2, bit a bit). El periodo máximo que puede producir esta secuencia es $P = 256\,x\,(2^{31}\text{-}1)(2^{29}\text{-}1)(2^{61}\text{-}1) = 6{,}8056\ 10^{38}$.

El reloj CLK_2 corresponde a la velocidad a la que llegan los bits de sonido y datos cifrados. Por tanto, el generador J_3 debe producir los bits pseudoaleatorios a esta velocidad, que es de 1,546875 MHz. Como esta información digital llega en forma de paquetes de 751 bits de longitud, también se puede decir que el flujo binario es de 2.050 paquetes/seg.

Pero, en cada paquete, hay 23 bits que corresponden a la cabecera (PH) que llegan en claro, por lo que no debe ser descifrada. Esto se consigue usando una señal de validación (VAL), que consiste en un pulso de 31 ciclos de reloj que se repite cada vez que pasa la cabecera de un paquete. Es decir, a una tasa de 2.050 veces/seg. o, lo que es lo mismo, cada 487,8 microsegundos. Durante estos 23 ciclos de reloj, en que pasa la cabecera de cada paquete, el generador J_3 no produce secuencia de bits pseudoaleatoria. Este intervalo lo aprovecha el generador J_1 para producir 61 bits que carga como estado inicial del generador J_3 (lo que implica que el generador J_1 debe producir bits a una tasa de 125.050 bits/seg). Por tanto, el estado inicial del generador J_3 cambiará cada 487 microsegundos y, por tanto, generará una secuencia distinta para cada paquete.

Para aplicar un criptoanálisis usando el test de consistencia lineal a este generador se necesitarán: $N > m + n2^h = 29 + 29\,x\,2^5 = 957$ bits sobre los que habrá que aplicar $2^{m+h} = 2^{31} = 1{,}718\ 10^{10}$ tests de consistencia lineal en menos de 487 microsegundos. Esto es claramente imposible. Por tanto, podemos concluir que el sistema de gestión de claves protege al sistema contra ataques criptoanalíticos basados en el test de consistencia lineal [CRU93].

B) Generador para el cifrado de redes de banda ancha ATM

Este generador de secuencias pseudoaleatorias ha sido propuesto [FUS91] para realizar un sistema de protección criptográfica de datos basado en el cifrado en flujo en el proyecto PLANBA (Plan Nacional de Banda Ancha) . Las elevadas velocidades que pueden llegar a implicar este tipo de redes, hacen necesaria la utilización de este tipo de técnicas de cifrado en flujo debido al pequeño retardo que implican, ya que la información cifrada se obtiene mediante la suma módulo 2 entre la secuencia de información y la secuencia producida por el generador de bits aleatorios.

La estructura básica de este generador consiste en un LFSR de L etapas y polinomio de realimentación primitivo, de forma que la secuencia de salida tenga un periodo de $2^L\text{-}1$. L se eligirá de forma tal que cumpla $L = n+2^n$, si L y n son enteros.

El funcionamiento de este generador es el siguiente: las n primeras celdas del LFSR, empezando por la izquierda, constituyen los bits de rotación R_j ($j = 0, 1,..., n\text{-}1$). El valor que en cada instante tomen estos n bits de rotación determinará la cantidad de veces que se deberá aplicar una

rotación hacia la derecha sobre las restantes 2^n celdas del LFSR, los bits rotantes, que se denominarán X_i (i = 0, 1,..., 2^n-1). La rotación que se realice sobre los bits rotantes deberá ser cíclica. Los nuevos 2^n bits obtenidos de la rotación se almacenarán en un nuevo registro (posición de memoria) y se denotarán por Y_i (i = 0, 1,..., 2^n-1).

Este proceso de rotación múltiple (RM) introduce un filtrado no lineal sobre el LFSR, de forma que a cada estado de L bits se le hace corresponder una secuencia de 2^n bits rotados:

$$RM: (R_0, R_1, \ldots, R_{n-1}, X_0, X_1, \ldots, X_{2^n-1}) \rightarrow (Y_0, Y_1, \ldots, Y_{2^n-1})$$

Por tanto, a partir de esta estructura se pueden obtener $M = 2^n$ secuencias de bits rotados $\{Y_0\}, \{Y_1\}, \ldots, \{Y_{2^n-1}\}$ todas ellas de periodo $P = 2^n$-1. Para cualquiera de estas 2^n secuencias $\{Y_i\}$, la función booleana que permite obtener un nuevo bit de la secuencia vendrá expresada por:

$$Y_i = c_i \oplus a_0 R_0 \oplus \ldots \oplus a_{n-1} R_{n-1} \oplus a_{01} R_0 R_1 \oplus \ldots \oplus a_{n-2,n-1} R_{n-2} R_{n-1} \oplus \ldots \oplus a_{01..n-1} R_0 R_1 \ldots R_{n-1} \qquad (1)$$

donde $c_i = X_i$, $a_{01..n-1} = X_0 \oplus X_1 \oplus \ldots \oplus X_{n-1}$, y los restantes coeficientes $a_{ij\ldots}$ corresponden a funciones OR-exclusiva de los bits rotantes X_i. La complejidad de las 2^n secuencias obtenidas, se pueden acotar mediante la expresión:

$$C(Y_i) \leq \sum_{k=0}^{\lambda} \binom{L}{k}$$

que alcanzará la igualdad, o bien estará siempre muy próxima a ella. λ es el orden de la función generatriz de cada una de las secuencias $\{Y_i\}$ y vale $\lambda = n+1$.

La ecuación (1) incluye un término lineal y términos de órdenes pequeños, por lo que las secuencias obtenidas presentarán una buena distribución de unos y ceros. Otra propiedad interesante para aplicaciones de cifrado en flujo será la buena inmunidad a la correlación que presenta, ya que las 2^n secuencias obtenidas presentan una baja correlación con la secuencia original generada por el LFSR.

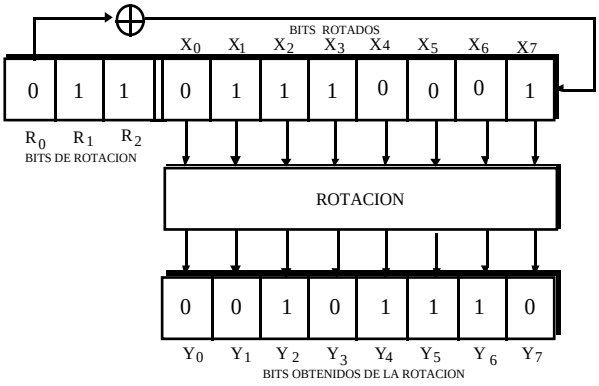


Fig. 7.10 Cifrador para redes ATM

En la figura 7.10 se puede ver un ejemplo de este generador, donde se ha elegido un LFSR con $L = 3+2^3= 11$ celdas y, por tanto, de periodo $P = 2^{11}-1= 2.047$ bits. Contiene pues 3 bits de rotación (R_0, R_1 y R_2) que se corresponden con las tres celdas de la izquierda del LFSR y de 8 bits rotantes (X_0,..., X_7), que corresponden con las celdas de la derecha. Los tres bits de rotación pueden determinar 8 posibles desplazamientos para los 8 bits rotantes. En el ejemplo, estos bits rotantes toman el valor decimal 3, por lo que se rotan cíclicamente los bits 3 veces hacia la derecha, obteniéndose la secuencia Y_0,.., Y_7. Cada una de las celdas correspondientes a los bits rotados, se puede además considerar como una salida, con lo que este dispositivo podrá generar 8 secuencias $\{Y_0\}$, $\{Y_1\}$,..., $\{Y_7\}$, todas ellas de periodo $P = 2^{11}-1= 2.047$ bits.

Se puede generalizar la estructura no lineal descrita. Para ello, se puede considerar un registro de desplazamiento inicial con un número de etapas tal que englobe a varios rotadores adyacentes, ya sean de la misma o diferente longitud. Además, se puede hacer que las rotaciones para los diferentes rotadores sean a derechas, a izquierdas o combinaciones de ambas. La ventaja de encadenar rotadores es que el periodo de las secuencias generadas $\{Y_i\}$ será mayor, debido a la mayor longitud del LFSR de partida, pero se mantendrán las propiedades que tenía cada rotador por separado.

Además, se pueden combinar las secuencias producidas mediante una técnica de plegamientos. Si se han obtenido dos secuencias $\{X_i\} = x_0, x_1,..., x_{n-1}$ e $\{Y_i\} = y_0, y_1,..., y_{n-1}$, mediante este generador, el plegamiento consistirá en la suma módulo $2n$ entre ambos números x ($x = x_0x_1...x_{n-1}$) e y ($y = y_0y_1...y_{n-1}$), con lo que se obtiene la secuencia $z = x+y$ ($z = z_0z_1...z_{n-1}$) donde cada bit se obtiene como $z_i = x_i \oplus y_i \oplus c_i$ (i = 0, 1, ..., n-1), y donde c_i representa el acarreo proveniente de la suma de los bits anteriores al bit considerado y con valor $c_i = c_{i+1}x_{i+1} \oplus c_{i+1}y_{i+1} \oplus x_{i+1}y_{i+1}$ (i = 0, 1,..., n-2) y $c_{n-1}= 0$.

C) Cifrado en comunicaciones móviles GSM

En 1982 se creó un grupo de estandarización dentro del CEPT (*Conférence Européene de Postes et Telécommunicacions*) para especificar un sistema europeo único de radiocomunicaciones en la banda de 900MHz. Finalmente, en 1991 se pudo hacer la primera demostración pública del sistema de telefonía celular digital que pasó a denominarse GSM (*Global System for Mobile Communications*) [MOU93]. Una de las múltiples ventajas que presenta este sistema frente a los tradicionales sistemas analógicos, y es la que interesa describir aquí, es que introduce funciones de seguridad [ETS90][ETS91] para proteger a la red frente a accesos fraudulentos y garantizar la privacidad de los usuarios. Estos servicios, que se analizarán en este apartado debido a que hacen uso de secuencias o números pseudoaleatorios, son la autentificación del usuario (para evitar el acceso a la red de usuarios no autorizados), el cifrado de la información del usuario en el radioenlace y de algunos elementos de señalización (para prevenir escuchas por parte de terceros), y la protección de las identidades de los usuarios (para imposibilitar el seguimiento de su localización por parte de terceros). Veamos cómo se realizan estos procesos y cómo hacen uso de las secuencias o números aleatorios [REC94].

C1) El proceso de autentificación

La prevención de accesos no autorizados se consigue mediante un proceso de autentificación de los usuarios. Por un lado, el usuario debe identificarse frente al SIM (el módulo de identificación de usuario) tecleando su número de identificación personal (PIN).

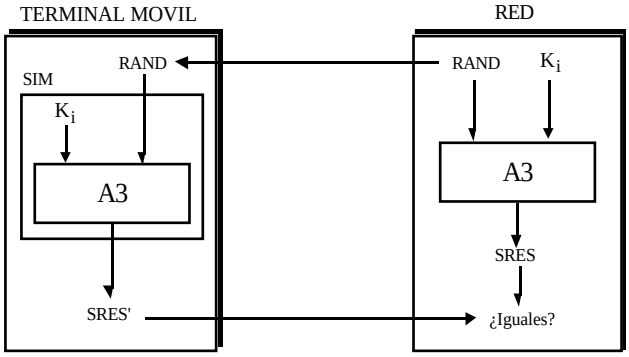


Fig. 7.11 Proceso de autentificación en GSM

El SIM consiste en una targeta inteligente estándar con la que se consigue independizar el terminal de la subscripción realizada por el abonado. Una vez hecho esto, la red comprobará la validez del SIM mediante un proceso de autentificación. Para realizar este proceso, la red le realiza una pregunta al SIM que sólo él debe ser capaz de contestar. La forma en que se realiza este protocolo de autentificación puede verse en la figura 7.11. Cada usuario dispone de una clave secreta de autentificación K_i que sólo es conocida por su SIM y por la red. El procedimiento es el siguiente. La red debe generar un número aleatorio de 128 bits (que se denomina RAND) y se lo envía al terminal. Además lo cifra mediante un algoritmo de cifrado denominado A3, utilizando K_i como clave. El resultado SRES de 32 bits es comparado con la respuesta SRES` recibida del terminal. En caso de ser ambos iguales se da por auténtica la identidad del usuario. Los detalles de implementación del algoritmo A3, así como la longitud de la clave K_i se dejan a la opción del operador de red, pero siempre que garantice la interoperabilidad con otros operadores.

Para garantizar un buen nivel de seguridad, el algoritmo A3 debe ser una función de un solo sentido, lo que quiere decir que una vez conocidos el número pseudoaleatorio RAND y la clave K_i es fácil calcular el número SRES, pero conociendo SRES y RAND será difícil el cálculo de K_i.

C2) El proceso de cifrado

Una de las ventajas del GSM es que permite la confidencialidad de la información de los usuarios sobre el radioenlace, consiguiéndose además niveles de seguridad muy buenos. La figura 7.12 muestra la ubicación del proceso de cifrado dentro de la cadena de transmisión del canal digital de comunicación establecido.

En GSM se utiliza un cifrado en flujo, es decir, cada bit se cifra de forma independiente, lo que evita que se produzca propagación de errores.Tanto la operación de cifrado como la de descifrado se realizan aplicando una operación OR exclusiva bit a bit entre los 114 bits codificados de un *time slot* y otros 114 bits generados por un generador de secuencias pseudoaleatorias denominado A5, el cual es común a todos los operadores GSM, aunque sus detalles de implementación son guardados en secreto por diseñadores y fabricantes.

Para generar la secuencia de cifrado para cada *time slot*, el algoritmo A5 utiliza dos datos de entrada: el contador de trama y una clave de sesión K_c, obtenida a través del algoritmo A8 cuando a la entrada están K_c y RAND.

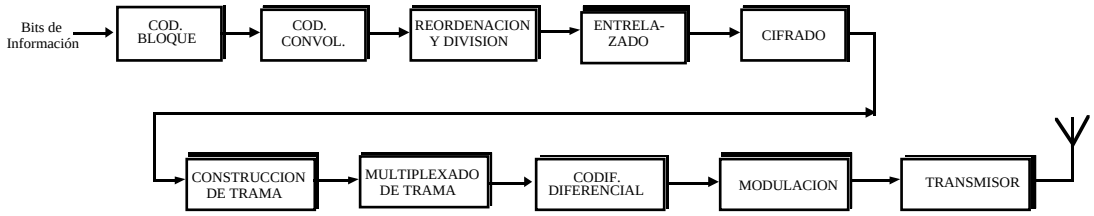


Fig. 7.12 Diagrama de bloques de un transmisor GSM

Utilizando el identificador de trama para sincronizar el generador A5, se consigue que el procesamiento de cada trama sea independiente y que la posible pérdida de una de ellas no afecte al resto.

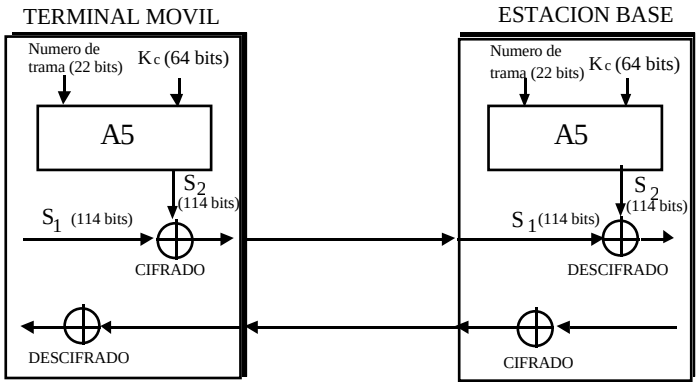


Fig. 7.13 Proceso de confidencialidad en GSM

Los canales de subida y bajada utilizan secuencias de cifrado distintas, debido a que los números de trama son diferentes en ambos. La figura 7.13 muestra el principio de funcionamiento de este tipo de cifrado que hace uso de una secuencia pseudoaleatoria.

C3) La gestión de claves

Cuando se establece una conexión, hay dos posibilidades para determinar cuál va a ser la clave de cifrado que se va a usar. La primera, es crear una nueva clave, y la segunda, es usar la última clave negociada. La generación de una nueva clave de cifrado se hace durante el proceso de autentificación utilizando un algoritmo denominado A8 que toma como entradas la clave K_i y el número pseudoaleatorio RAND, tal como muestra la figura 7.14. El algoritmo A8 no está especificado y se deja a la opción del operador. En realidad, en la práctica ambos algoritmos pueden ser el mismo. Puede utilizarse, por ejemplo, un algoritmo que, utilizando como entradas RAND y K_i, genere 92 bits, de los cuales 32 formarán SRES y los 64 restantes K_c. Dado que el algoritmo A5 requiere 64 bits de clave, el resto se rellenan con ceros. En el futuro se podrá aumentar el nivel de seguridad aumentando tan sólo la longitud real de la clave, sin tener que modificar el algoritmo A5.

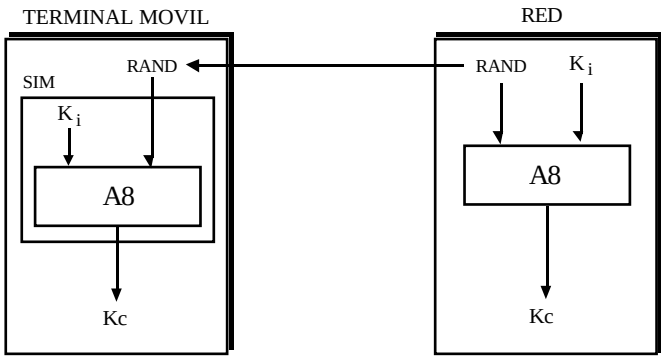


Fig. 7.14 Gestión de claves en GSM

7.2 La dispersión de energía en comunicaciones

Esta técnica se utiliza ampliamente en comunicaciones de datos en enlaces vía satélite y su motivación está en conseguir que la energía de la señal transmitida se mantenga dispersa a lo largo de todo el ancho de banda del canal de transmisión, además de mantener un espectro lo más constante posible [GIT92].

Por ejemplo, la transmisión se está realizando a una cierta cantidad de Mbits por segundo que ocupa un cierto ancho de banda, si se da el caso de que durante ciertos intervalos de tiempo la información que se transmite está constituída sólo por ceros o sólo por unos, o por subsecuencias más o menos largas de éstos, el ancho de banda de la señal transmitida podría colapsarse debido a la mayor distancia entre las transiciones que se producirán al pasar de cero a uno o viceversa.

Esto no sólo representará un problema en cuanto a la forma del espectro, sino que también puede producir una pérdida de sincronización en el receptor debido a la pérdida de transiciones y también para los ecualizadores del receptor.

Para solucionar este problema, se usa una técnica de aleatorización de la señal, de forma que las secuencia de información se combinan, módulo 2, con una secuencia pseudoaleatoria. El resultado de esta combinación es otra secuencia con forma pseudoaleatoria que se enviará al canal y que resolverá los problemas anteriores ya que, debido a su nueva naturaleza, tendrá transiciones uniformemente distribuídas, con lo que mantendrá un espectro bastante constante en cuanto a anchura de banda y amplitud y permitirá una correcta sincronización y funcionamiento de los ecualizadores en el receptor. En el receptor, se deberá realizar el proceso inverso (desaleatorización), volviendo a repetir la suma módulo 2 entre la señal aleatorizada entrante y la misma secuencia pseudoaleatoria que se usó en el transmisor.

Otra consideración que debemos hacer con respecto a esta aplicación es que la secuencia pseudoaleatoria se debe generar a la misma velocidad que la información que vamos a transmitir, es decir, se debe combinar un bit de información con un bit generado por el generador pseudoaleatorio. Es importante distinguir esto ya que, tal como veremos más adelante, existen otras aplicaciones como las modulaciones *spread spectrum* en las que se realiza un proceso similar a éste, pero combinando cada bit de información con diversos bits pseudoaleatorios (que tienen un periodo menor), con lo que se expandirá el espectro de la señal de información.

Volviendo a la aplicación que nos ocupa, hay que observar que, si bien tecnológicamente el proceso descrito es similar al que se realizaba en el cifrado en flujo, en este caso la motivación final es distinta, y por tanto las características que se van a requerir de las secuencias pseudoaleatorias empleadas también lo van a ser. Por ejemplo, en las aplicaciones de cifrado en flujo vimos que la complejidad de las secuencias pseudoaleatorias era un aspecto de vital importancia, ya que la información enviada al canal debía ser impredictible para evitar posibles ataques a la seguridad.

En esta aplicación, el factor de elevada complejidad no tiene especial interés, ya que la finalidad del proceso es tan sólo aleatorizar la secuencia y no protegerla contra revelaciones a intrusos, por lo que bastará utilizar generadores que muestren un buen comportamiento estadístico en las secuencias pseudoaleatorias generadas sin que sea necesario que las secuencias que producen muestren gran complejidad.

Esto nos lleva a una elección clara, que son los registros de desplazamiento con realimentación lineal que son, de hecho, los generadores que se usan en la práctica para realizar esta aplicación. Se analizarán dos ejemplos prácticos en sistemas reales que realizan este proceso: el sistema de transmisión de televisión vía satélite MAC/paquetes y el nuevo estándar de transmisión de televisión digital de televisión DVB (*Digital Video Broadcasting*).

7.2.1 La aleatorización en el sistema MAC/paquetes

Este sistema, que ya hemos visto en apartados anteriores debido a que utiliza un sistema de cifrado en flujo para proteger la información que se transmite, también realiza un proceso de aleatorización de la señal para conseguir los objetivos anteriores [EBU86].

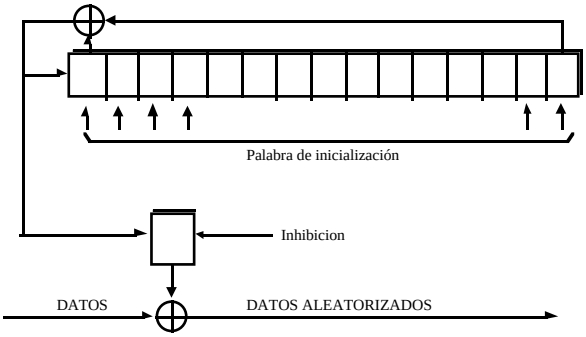


Fig. 7.15 Aleatorizador del sistema MAC/paquetes

La aleatorización para dispersar la energía se realiza después del entrelazado de bits y antes del modulador. Para ello, se utiliza un LFSR como el que muestra la figura 7.15 que actúa a una velocidad de 20.25 MHz. para los estándares C y D-MAC, y a 10.125 MHz. para el estándar D2-MAC. Puesto que la aleatorización no debe actuar en ciertas partes de las ráfagas producidas por estos sistemas (por ejemplo no debe actuar en los primeros 7 bits de cada ráfaga de datos en los sistemas C y D, o en los primeros 6 bits en el sistema D2 ni en las líneas 624 y 625), en estos intervalos el LFSR seguirá

produciendo su secuencia de salida pero su acción será inhibida mediante una puerta. Para los sistemas C y D se aleatorizan bloques de 123.354 bits sumándolos módulo 2 con la secuencia de salida del LFSR. El LFSR tiene 15 celdas y polinomio primitivo, con lo que tendrá un periodo $P = 2^{15}-1 =$ 32.767 bits, y se inicializará cada 652 líneas de televisión.

7.2.2 La aleatorización en el sistema de transmisión digital de televisión DVB

Este es un nuevo estándar de transmisión de televisión [ETS94][COM93][GAV96] que, a diferencia del sistema MAC/paquetes que transmite la imagen en forma analógica, y el sonido y los datos en forma digital, permitirá transmitir toda la información (imagen, audio y datos) en forma completamente digital y comprimida, utilizando la compresión MPEG2. La figura 7.16 muestra el diagrama de bloques del transmisor de este sistema, donde se puede ver el bloque de aleatorización [ISO94].

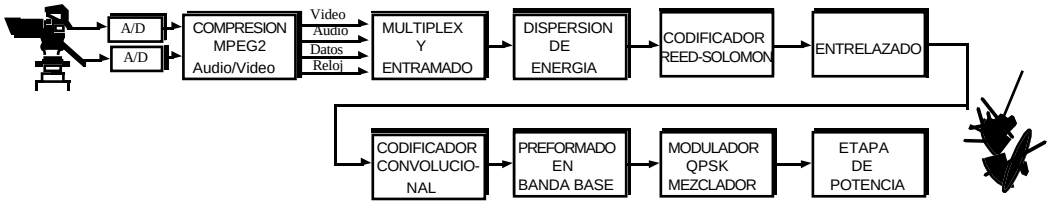


Fig. 7.16 Transmisor DVB

Por otra parte, la figura 7.17 muestra el LFSR que se utiliza en este sistema para realizar este proceso. Tal como se puede ver, tiene 15 celdas y polinomio primitivo de realimentación, con lo que su periodo será de $P = 2^{15}-1 = 32.767$ bits.

La información que se transmite en este sistema está constituída por paquetes de longitud fija de 188 bytes, que contituyen lo que se llama la *trama de transporte MPEG-2*. Este paquete contiene además un byte de sincronismo (por ejemplo 47_{HEX}), tal como muestra la figura 7.18. De acuerdo con el estándar DVB, estos paquetes (que contienen la información comprimida MPEG2) deben aleatorizarse, utilizando para ello el generador de secuencias pseudoaleatorias de la figura 7.17, cuyo polinomio de realimentación es $f(t) = 1 = t^{14}+t^{15}$.

La inicialización de dicho generador debe realizarse cargando en sus celdas la secuencia "100101010000000" al comienzo de cada 8 paquetes de transporte. Para suministrar una señal de inicialización al aleatorizador, el byte de sincronismo del primer paquete de cada grupo de 8 debe invertirse (de 47_{HEX} pasaría a $B8_{HEX}$). A este proceso se le denomina *adaptación del multiplex de transporte* y se muestra en la figura 7.19. El primer bit a la salida del generador se deberá aplicar sobre el primer bit (el más significativo) del primer byte que siga al byte de sincronismo MPEG2 invertido (por ejemplo $B8_{HEX}$).

Para permitir otras funciones de sincronización, durante los 7 bytes de sincronismo de los 7 paquetes de transporte restantes, la generación de bits por parte del generador no se detendrá, pero su combinación con estos bytes de sincronismo se inhibirá y los dejará sin aleatorizar. Por tanto, el periodo de la secuencia será de 1.503 bytes.

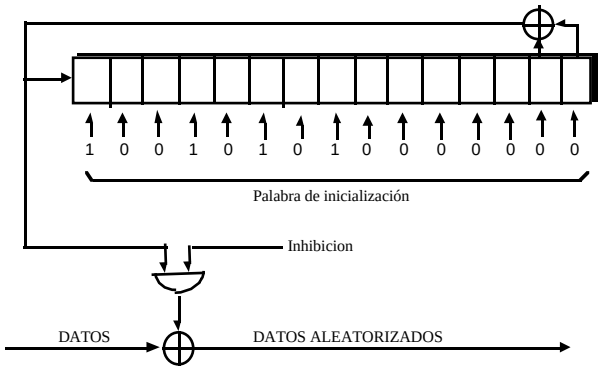


Fig. 7.17 Aleatorizador del estándar DVB

El proceso de aleatorización en este sistema debe permanecer activo incluso cuando no existan datos de entrada o cuando ésta no cumpla con el formato de la trama de transporte MPEG2 (1 byte de sincronismo + 187 bytes), lo que evitará la emisión de una portadora no modulada.

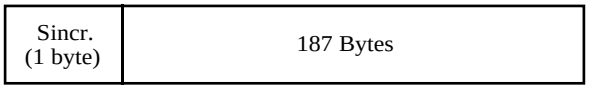


Fig. 7.18 Paquete del multiplex de transporte MPEG2

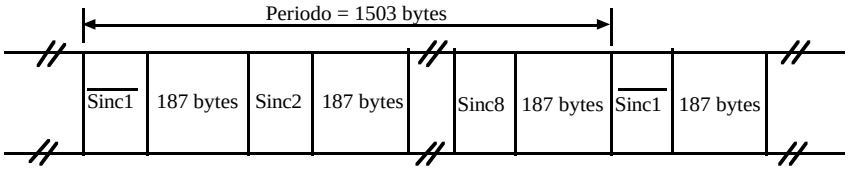


Fig. 7.19 Paquetes de transporte aleatorizados

7.3 La simulación de procesos

Debido a la creciente complejidad de los sistemas actuales, el hecho de ser capaces de cuantificar sus prestaciones en la fase de su diseño se convierte en un punto de vital importancia, ya que permitirá elegir entre las diversas alternativas posibles. Es necesario realizar un modelo de simulación del sistema que, previa validación, se pueda modificar y evaluar de nuevo las prestaciones del sistema simulado [GRAYB].

El concepto de simulación se basa en la modelización de sistemas y, si bien este último concepto tiene muchos años de existencia, tan sólo en los últimos años se ha podido aplicar la simulación debido a la aparicion de los ordenadores.

Sannon definió [SHA75] la simulación como el proceso consistente en diseñar un modelo computerizado de un sistema (o proceso) y de realizar experimentos sobre este modelo con los propósitos tanto de comprender su comportamiento como para evaluar diversas estrategias para la operación del mismo. Toda simulación se basa en una metodología consistente en 4 fases denominadas planificación, modelización, validación (comprobación que el modelo es una representación correcta del sistema real bajo estudio) y aplicación, de forma que cada una de ellas se corresponde con cada uno de los 4 pasos del método científico (observación del sistema, formulación de hipótesis o teorías, predicción del comportamiento futuro bajo la suposición de que las hipótesis son correctas y, finalmente, comparación del comportamiento previsto con el actual).

Aunque los ordenadores que se emplean para realizar dichas simulaciones son completamente deterministas, los análisis que realizamos sobre los sistemas que modelamos requieren, a menudo, el uso de la teoría de la probabilidad debido a que muchos de los procesos implicados son inherentemente aleatorios. Por tanto, en cualquier simulación se deberán asignar ciertos valores a algunos eventos cuyo comportamiento parece seguir pautas aleatorias.

Este es uno de los problemas más comunes al intentar modelar sistemas reales, ya que muy pocos muestran un comportamirnto constante o predictible. Además, las observaciones y medidas que realicemos sobre él dependerán casi siempre del instante en que se realicen (entre otras cosas) y estas variaciones casi siempre suelen mostrar un comportamiento aleatorio.

Sin embargo, a pesar de la impredictibilidad inherente en estos sistemas, necesitamos un modelo o estructura matemática para describir su comportamiento. Para ello el sistema se describe a través de un modelo probabilístico que en la mayoría de los casos se obtendrá a través de la experimentación (realizando experimentos sobre el sistema, tomando como datos las salidas que muestra el sistema a esos experimentos y desarrollando un modelo en función de esos datos). Una vez hecho esto se pueden realizar inferencias basadas en ese modelo y realizar experimentos adicionales para validar el modelo. En general, cuantos más experimentos se realicen sobre el sistema, más exactamente reflejará su verdadera naturaleza.

A la hora de experimentar con un modelo de simulación, se suelen emplear variables aleatorias para representar ciertos aspectos del modelo que se quiere simular y deben seguir una determinada distribución estadística.

En el mejor de los casos, si esta distribución es conocida se simplificará mucho el trabajo. Si no es así, se deberá obtener esta distribución, empleando alguna de las diversas técnicas existentes para caracterizar el comportamiento de una variable aleatoria estudiando para ello un conjunto de muestras obtenidas mediante una serie de observaciones.

La mayoría de generadores de secuencias pseudoaleatorias para implementación por *software* estudiados en el capítulo 2 serán potencialmente útiles para realizar simulaciones de procesos que requieran como entrada un cierto grado de aleatoriedad.

7.4 La modulación *spread spectrum*

La difusión del espectro es una técnica de modulación que se ha venido desarrollando desde mediados de los años 50. Las aplicaciones iniciales de esta técnica fueron las comunicaciones tácticas militares, los sistemas de guíado y los sistemas de protección contra las perturbaciones debidas a la propagación multicamino. En este tipo de modulación, la señal que se transmite ocupa un ancho de banda superior al mínimo necesario para enviar la información, por lo que realiza el ensanchado del

espectro mediante un código que es independiente de los datos. En el receptor se usa el mismo código de forma síncrona para recuperar la señal de datos [SKAUG][PIC82].

La principal característica de las señales moduladas en *spread spectrum* es que el ancho de banda de la señal transmitida (W Hz), es mucho mayor que el de la información (R bits/s), introduciendo de esta forma una redundancia que permite suprimir los indeseables efectos de la interferencia inherente en las transmisiones digitales sobre enlaces de satélite. Un elemento de gran importancia en el diseño de sistemas de *spread spectrum* es el de la aleatoriedad, ya que la modulación consistirá en combinar la señal de información $d(t)$ con una señal pseudoaleatoria $s(t)$, tal como muestra la figura 7.20, de forma que la señal transmitida se parezca al ruido aleatorio y, por tanto, sea difícil de modular por receptores que no dispongan de la secuencia (o código) de modulación.

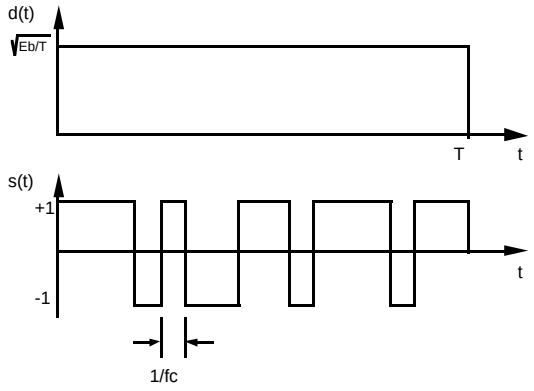


Fig. 7.20 Modulación spread spectrum en el dominio temporal

Tal como muestra dicha figura, la velocidad de la secuencia pseudoaleatoria $s(t)$ es mayor que la de la señal de datos $d(t)$, con lo que el resultado de la modulación implicará un ensanchado del espectro, tal como muestra la figura 7.21. Los principales objetivos al emplear esta técnica son:

1) Suprimir los efectos indeseables de posibles interferencias intencionadas (*jamming*), las interferencias debidas a otros usuarios del canal y la propia interferencia debida a la propagación multicamino. Para evitar la primera, es necesario que el atacante que pretenda interferir la comunicación no tenga conocimiento de las características de la señal que se va a emplear. Para conseguir esto, el transmisor debe introducir cierta aleatoriedad a la señal que se va a transmitir, pero teniendo especial cuidado en que ningún posible atacante pueda tener conocimiento de la secuencia pseudoaleatoria empleada.

La interferencia de otros usuarios aparece en sistemas de acceso múltiple en los que diversos usuarios comparten un mismo ancho de banda, de forma que en cualquier momento, cualquier subconjunto de estos usuarios pueda transmitir su información de forma simultánea y puedan ser distinguibles unos de otros por medio de combinar sus respectivos flujos de información con secuencias pseudoaleatorias (también llamadas códigos) diferentes y ortogonales. De esta forma, para una determinada comunicación, sólo el usuario autorizado que disponga del código correcto podrá recuperar la información que le corresponde. Finalmente, los componentes resultantes de la

propagación multicamino se pueden ver como una forma de autointerferencia, que también se podrá eliminar combinando las señales transmitidas con secuencias pseudoaleatorias.

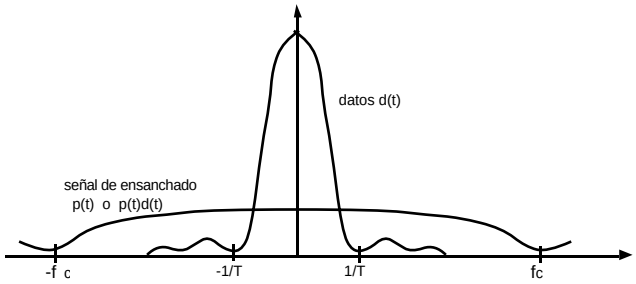


Fig. 7.21 Modulación spread spectrum en el dominio frecuencial

2) Ocultar la señal y hacerla difícil de detectar para usuarios no autorizados. Esto se puede conseguir ocultando los mensajes bajo el ruido de fondo mediante el ensanchado de su espectro y transmitiendo la señal resultante con una potencia media baja. A estas señales se las denomina de baja probabilidad de intercepción (LPI).

3) Se puede conseguir, además, privacidad de la comunicación combinando la señal transmitida con una secuencia pseudoaleatoria, de forma que sólo el usuario autorizado que disponga del código (clave) que se usó para producir la secuencia pseudoaleatoria pueda descifrar la señal entrante.

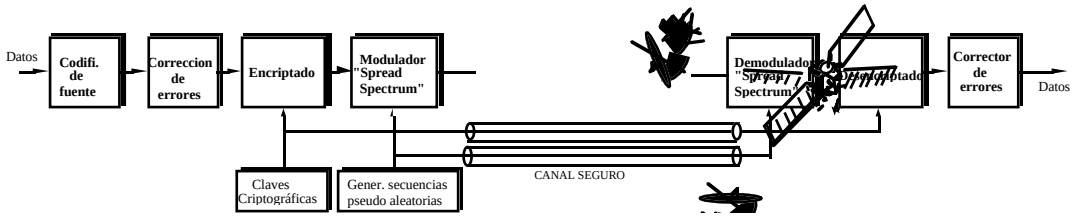


Fig. 7.22 Comunicación spread spectrum

La figura 7.22 muestra los bloques funcionales implicados en una comunicación *spread spectrum*. La idea básica de esta comunicación consiste en reemplazar cada bit de datos por una forma de onda codificada, de forma que sólo se pueda detectar de forma óptima en el receptor si se dispone del código que se utilizó para codificarla en el transmisor. Para este propósito, se necesitan dos generadores de secuencias pseudoaleatorias, uno en el modulador y otro en el demodulador, de forma

que estén sincronizados. Estos generadores producirán secuencias pseudoaleatorias binarias $s(t)$ que se combinarán en el modulador con la señal digital que se va a transmitir $d(t)$. En el receptor se deberá realizar el proceso inverso empleando la misma secuencia. La figura 7.23 muestra como el modulador y el demodulador emplean esta secuencia pseudoaleatoria. En esta figura se puede ver que en el transmisor se usan 2 secuencias pseudoaleatorias que se multiplican con la secuencia de datos, de forma que la velocidad de la secuencia pseudoaleatoria es mucho mayor que la velocidad de ésta, con lo que se expandirá su espectro.

En este modulador [VIT94], las dos formas de onda obtenidas modulan en fase y cuadratura una portadora y, una vez sumadas estas dos componentes, se transmiten al canal. El receptor realiza el proceso inverso suprimiendo las secuencias pseudoaleatorias de ensanchado antes de realizar el procesado en banda base.

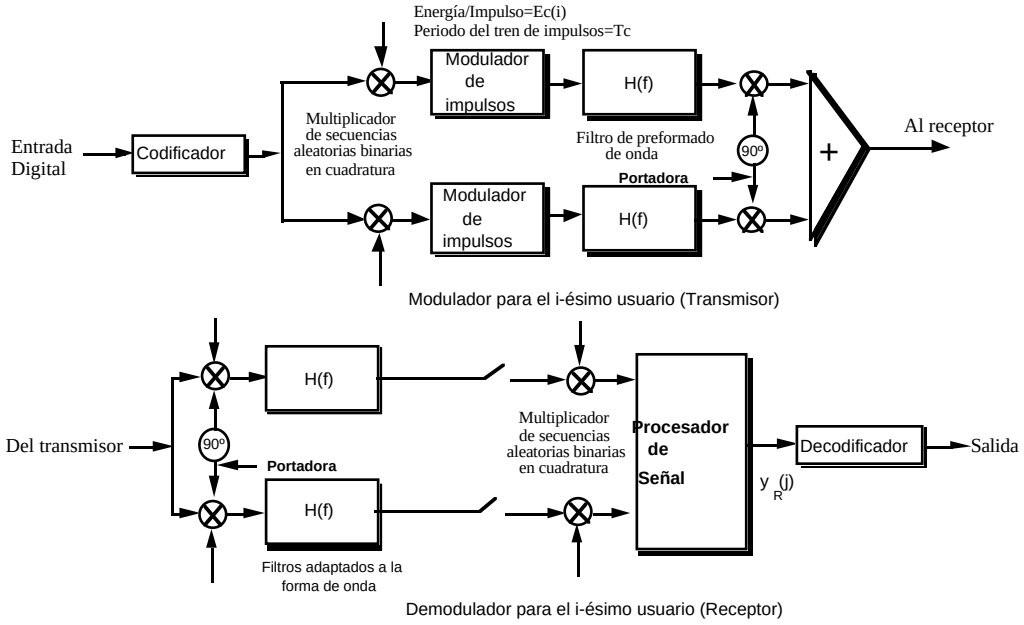


Fig. 7.23 Modulador spread spectrum

7.4.1 Técnicas de acceso múltiple por división de código (CDMA)

Una aplicación importante de la modulación *spread spectrum* está en que varios usuarios comparten de forma simultánea el ancho de banda de un canal de comunicaciones. Para ello, a cada usuario se le asigna un código pseudoaleatorio distinto. A este tipo de técnica de acceso se le denomina acceso múltiple por división de código (CDMA) y cada secuencia pseudoaleatoria identificará de forma única al usuario [BAM93][BAM92]. Por ejemplo, si se asigna al usuario 1 la secuencia s_1, al usuario 2 la secuencia s_2, etc, un receptor que desee recibir la comunicación con el usuario 1 recibirá en su antena la energía emitida por todos los usuarios (que será vista como ruido *cross-talk*). Sin embargo, después de demodular la señal recibida empleando la secuencia s_1, captará

toda la energía del usuario 1 y una pequeña porción de la energía de los otros usuarios. Por tanto, para que esta técnica de acceso múltiple sea efectiva, se debe asignar un código a cada usuario y el requisito fundamental será que cada uno de estos códigos esté incorrelado con los códigos usados por los otros usuarios que comparten el mismo canal. Si se cumple esta propiedad, el proceso de demodulación en el receptor rechazará las transmisiones de los usuarios no deseados y la interferencia (incluídos los productos de intermodulación). El grado de rechazo de las señales incorreladas se conoce con el nombre de ganancia de procesado y es también la relación entre el ancho de banda de la secuencia antes y después de la modulación.

La implementación más común de sistemas CDMA es la llamada de secuencia directa (DS-CDMA), en la que el transmisor multiplica su señal de datos por un código de ensanchado de espectro bipolar pseudoaleatorio que normalmente presenta una tasa entre 1 Mchip/s y 10 Mchips/s (el término chip se usa para representar un bit del código de ensanchado de espectro). En el receptor se debe multiplicar la señal recibida del canal, previa sincronización, por el mismo código de ensanchado de espectro, con lo que se restaurará el ancho de banda de la señal de datos original. Esta doble multiplicación se puede ver como dos inversiones de fase o también como dos sumas módulo 2.

Si b_i representa el bit i-ésimo de la secuencia pseudoaleatoria, y c_i representa el correspondiente bit de la secuencia de datos, la suma módulo 2 se representará como: $a_i = b_i \oplus a_i$. Entonces el modulador deberá realizar una aplicación entre la secuencia $\{a_i\}$ y una señal modulada en fase (PSK o *Phase Shift Keying*) de la forma:

$$s(t) = \pm Re\left[u(t)e^{j2\pi f_c t}\right] \qquad \text{donde:} \quad \begin{cases} si \quad a_i = 0 \quad entonces \quad u_i(t) = u\left(t - iT_c\right) \\ si \quad a_i = 1 \quad entonces \quad u_i(t) = -u\left(t - iT_c\right) \end{cases}$$

y $u(t)$ representa a un pulso de duración T_c segundos (y por tanto frecuencia $f_c = 1/T_c$) y forma arbitraria. Por tanto, se realiza una aplicación entre la secuencia de datos binaria con elementos {0,1} y otra secuencia secuencia formada por pulsos positivos y negativos o, lo que es lo mismo, se transforma la secuencia de datos en otra cuyos elementos son {-1, 1} y que se llama secuencia bipolar ya que corresponde a la anterior secuencia de pulsos positivos y negativos.

Aunque la técnica DS-CDMA no es, al menos teóricamente, tan eficiente espectralmente como otras técnicas de acceso múltiple como son la FDMA (acceso múltiple por división en frecuencia) o la TDMA (acceso múltiple por división en el tiempo), en la práctica se consigue con ella una reducción de los niveles de intermodulación tal que permite conseguir que un mayor número de usuarios compartan simultáneamente el misno canal de banda limitada que con las dos técnicas anteriores. Sin embargo, el precio a pagar es un encarecimiento y aumento de complejidad en los equipos y la dificultad en conseguir una adecuada sincronización en el sistema receptor.

Otro tipo de sistema CDMA es el de saltos de frecuencia (FH-CDMA de inglés *frequency hopping*), en el que el ancho de banda disponible del canal se subdivide en un gran número de porciones continuas de frecuencia (*slots*), de forma que en cada instante, la señal a transmitir ocupe uno o más de los *slots* disponibles elegidos aleatoriamente. La modulación más usual en este tipo de sistemas es la *M*-aria PSK. Por ejemplo, si se utiliza una modulación con $M = 2$, se eligirá entre uno de dos posibles *slots* según el valor que tome en ese instante la secuencia pseudoaleatoria. La señal FSK resultante se trasladará por tanto en frecuencia en una cantidad determinada por la salida del generador pseudoaleatorio. Evidentemente, habrá que coger los bits de la secuencia pseudoaleatoria de

m en *m* si se quieren conseguir 2^m-1 posibles traslaciones de frecuencia. Aunque las técnicas DS y FH-CDMA son las más comunes en la práctica, existen otras técnicas para introducir aleatoriedad a la señal de datos como la de salto en el tiempo (TH del inglés *time hopping*) en la que un intervalo de tiempo (que es mucho mayor que el recíproco de la tasa de la secuencia de datos), se subdivide en un gran número de pequeños *time slots*.

En este caso, los símbolos de información codificados se transmiten en un determinado *time slot* elegido aleatoriamente y se realiza posteriormente una modulación PSK. También obtener sistemas híbridos entre las técnicas DS, FH y TH.

A) Requisitos de diseño de las secuencias pseudoaleatorias para aplicaciones CDMA

En el diseño de sistemas CDMA, será de vital importancia obtener familias de códigos que muestren muy baja correlación cruzada entre ellos, a fin de minimizar la interferencia entre pares de usuarios [PUR77][SID71][SARWA]. Tradicionalmente se han empleado *m*-secuencias para conseguir estas familias. Sin embargo, su uso queda bastante restringido debido al limitado número de secuencias con baja correlación que se pueden conseguir. Ello ha llevado a utilizar otro tipo de técnicas como las secuencias de Gold [GOL68], las secuencias de Kasami y las funciones de Bent.

Debido al efecto que muestran los códigos en las prestaciones globales de una sistema de estas características, se deben hacer además otras consideraciones a la hora de su diseño [PUR74]:

1) Los códigos utilizados deben ser periódicos. Este es un requisito práctico, ya que si no fueran de longitud finita, si se perdiera la sincronización no sería posible recuperarla.

2) Para realizar correctamente dicha sincronización, los códigos deben mostrar un comportamiento impulsivo en su función de autocorrelación.

3) Para disminuir la posibilidad de un enganche falso, los lóbulos laterales de la función de autocorrelación no periódica deben ser de pequeño valor.

4) La correlación cruzada periódica entre los diferentes códigos debe ser pequeña para reducir la interferencia mutua entre los diferentes usuarios.

5) Las secuencias de códigos pseudoaleatorias deben mostrar un espectro de amplitud razonablemente constante por encima de algún valor de armónico.

B) Códigos de Gold

Gold [GOL67] describe un método para seleccionar *m*-secuencias con una cota superior determinada en la función de correlación cruzada. El resultado de Gold se basa en el siguiente teorema:

TEOREMA: Si θ es cualquier elemento primitivo de campos de Galois GF(2^m) y f_1 es el polinomio mínimo de θ, entonces sea f_1 el polinomio mínimo de q^t, donde $t = 2^{(m+e)/2} + 1$ con $e = 1$ si n es impar y $e = 2$ si n es par.

Si *k* es la *m*-secuencia generada por el polinomio f_1, y *r* es la *m*-secuencia generada por el polinomio f_t, la correlación cruzada entre las secuencias *k* y *r* tomará tres valores posibles y estará acotada por *t*. Así tendremos que:

$$k \in V(f_1) \quad y \quad r \in V(f_t) \quad \rightarrow \quad \left|\theta^{kr}\right| \leq t$$

Las dos *m*-secuencias *k* y *r* se llaman par preferente de *m*-secuencias. Aunque existen muchos pares de secuencias que satisfacen el teorema de Gold, no es posible, sin embargo, contruir familias grandes de *m*-secuencias que satisfagan la cota y, por tanto, es difícil diseñar sistemas de acceso múltiple CDMA de tamaño razonable.

Sin embargo, la importancia del teorema de Gold radica en el hecho de que una pareja preferente de *m*-secuencias puede producir familias grandes de secuencias de longitud no máxima con su correlación cruzada acotada. Puesto que las secuencias no son de longitud máxima, su autocorrelación no será una función bivalor. Sin embargo, el valor de dicha autocorrelación para los valores fuera de fase satisfará la cota del teorema de Gold. Así pues, flexibilizando algo la condición de la función de autocorrelación, podremos conseguir grandes familias de secuencias con su correlación cruzada acotada. Estas secuencias generadas a partir de parejas preferentes de *m*-secuencias se denominan códigos de Gold. El generador de códigos de Gold se puede construir de dos formas:

1) Utilizando un registro de desplazamiento cuya función de realimentación se obtenga del producto de los polinomios f_1 y f_t.

2) Sumando módulo 2 las salidas de los dos registros de desplazamiento de los dos LFSR con polinomios de realimentación f_1 y f_t respectivamente.

En ambos casos, se obtendrá una familia de 2^m+1 secuencias con un periodo de 2^m-1 y con autocorrelación y correlación cruzada acotadas por *t*. Podemos verlo mediante un ejemplo:

EJEMPLO 7.1: Consideremos los polinomios $f_1 = x^{10}+x^3+1$ y $f_t = x^{10}+x^9+x^8+x^6+x^3+x^2+1$. El generador de códigos de Gold puede generarse, bien mediante la multiplicación de ambos polinomios, resultado $f_1 f_t = x^{20}+x^{19}+x^{18}+x^{16}+x^{11}+x^8+x^5+x^2+1$ (figura 7.24a), o bien sumando módulo 2 las salidas de dos registros de desplazamiento que usen como polinomios de realimentación los polinomios f_1 y f_t respectivamente (figura 7.24b). Ambas soluciones darán una familia de 1.025 secuencias con una correlación cruzada entre ellas acotada por 65.

Ya se ha visto que la autocorrelación periódica (par) de las secuencias de Gold estará acotada por t. Sin embargo, en [MAS75], Massey y Uhran muestran otra cota, aunque esta vez para la autocorrelación no periódica (impar) de los códigos de Gold, que viene dada por la siguiente expresión:

$$\hat{\theta}^k \leq 2^{n-1} + 2^{\lfloor n/2 \rfloor} + 2$$

donde $\lfloor n/2 \rfloor$ denota la parte entera del número *n*/2. Para acabar con el estudio de los códigos de Gold, las gráficas 7.1a y 7.1b muestran los resultados obtenidos por M. Soriano en [SOR96] al

realizar un estudio sobre la autocorrelación y correlación cruzada impar respectivamente de códigos de Gold de periodo 3937, obtenidos a partir de dos registros de longitudes 5 y 7 respectivamente.

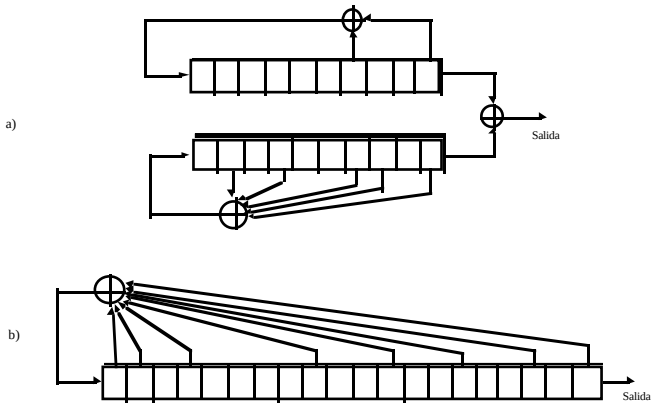
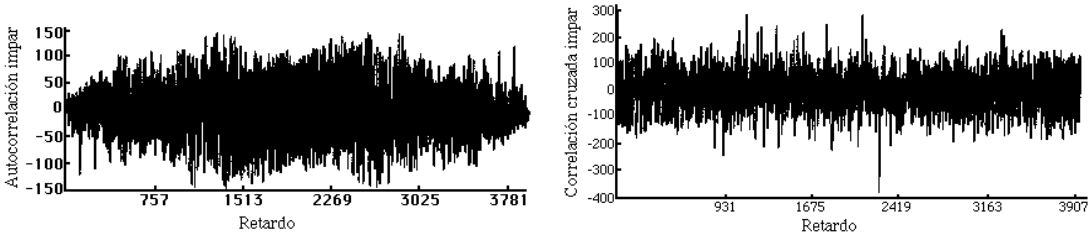


Fig. 7.24 Generador de códigos de Gold del ejemplo 7.1



a) b)

Gráfica 7.1 Autocorrelación y correlación cruzada aperiódica para secuencias de Gold de periodo 3937

7.5 Bibliografía

[ALA86] ALARD, M. *Distribution et Diffusion Terrestre en D2-MAC/packets (Terrestrial Distribution and Broadcasting of D2-MAC/packet signal*. Third International Conference on News Systems and Services in Telecommunications. Liège, November 1986.

[BAM92] *CDMA Field Test Results for Washington DC*. Bell Atlantic Mobile Systems. Rep. to FCC, Washington DC, Nov. 19, 1992

[BAM93] *CDMA Digital Cellular Technology Forum*. Beel Atlantic Mobile, Ameritech Mobile, GTE Mobilnet, US West NewVector and Qualcomm, San diego, Calif., Feb. 23-24, 1993.

[BEK82] BEKER; PIPER. *Cipher Systems*. John Wiley & Sons. New York 1982.

[BEK89] BEKER; PIPER. *Cryptography and Coding*. Clarendon Press. Oxford 1989.

[BLA82] BLASE, W.; HEINZMANN, P. *New Cryptographic Device with High Security Using Public Key Distribution*. IEEE Student Papers, 1982, pp. 145-153.

[BOY89] BOYAR, J. *Interferring Sequences Produced by Pseudo Random Number Generators*. J. ACM, Vol.36, Nº 1, pp. 129-141, Jan.1989.

[BRU82] BRUER, J. O. *On Nonlinear Combinations of Linear Shift Register Sequences.* Proc. IEEE Int. Symp, Jun. 1982, pp. 21-25.

[CAM78] CAMPBELL, C. M. *Design and Specification of Cryptographic Capabilities*. IEEE comm. Soc. Mag. Vol 16, pp. 15-19, 1978.

[CHEPY] CHEPYZHOV, V.; SMEETS, B. *On a Fast Correlation Attack on Certain Strem Ciphers.*

[COM93] COMINATTI, M.; MORELLO, A.; VISITIN, M. *Satellite Digital Multi-programme TV/HDTV*. M. EBU Review-Technical, Summer 93, EBU V4/MOD 235 Rev.

[CRU92] CRUSELLES, E.; MELÚS, J. L; PUIG, M. Evaluación del Generador de Jennings y su Aplicación en el Sistema de Acceso Condicional de la Norma MAC/packet. II Reunión Española sobre Criptología. Madrid, Octubre 1992.

[CRU93P] CRUSELLES, E. *Evaluación de la Seguridad del Sistems de Acceso Condicional Eurocrypt para la Protección de Transmisiones de Imagen, Voz y Datos Vía Satélite.* PFC. UPC. 1993.

[CRU93G] CRUSELLES, E.; MELÚS, J. L. *An Overview of Security in Eurocrypt Conditional Access System*. GLOBECOM´93. Houston, Texas. Noviembre 1993.

[CRU93M] CRUSELLES, E.; MELÚS, J. L. *Características de los sistemas de Acceso Condicional*. Mundo Electrónico, Marzo 1993.

[CRU94M] CRUSELLES, E.; MELÚS, J. L; BARBA, A; RECACHA, F. *Seguridad en Redes de Comunicaciones Móviles: Mecanismos y Servicios*. Mundo Electrónico, Abril 1994.

[CRU94C] CRUSELLES, E.; MELÚS, J. *Use of Supercomputing in Cryptography*. Curso High Performance Computing: Current Trends and Applications. Centro de Supercomputación de Cataluña, Barcelona, 21-22 de Noviembre de 1994.

[CRU96] CRUSELLES, E.; SORIANO, M.; J. L. MELÚS. *Spreading Codes Generator for Wireless CDMA Networks*. Wireless Personal Communications. Kluwer Academis Publishers. 1996.

[EBU86] *Specification of the System MAC/packet Family*. EBU document Tech. 3258 (1986), amenced by document spb 438.

[ETS90] *ETSI/TC GSM". Recommendation GSM 03.20: Security Aspects*. v. 3.0.1. January 1990.

[ETS91] *ETSI/TC GSM. Recommendation GSM 03.20. Security Related Network functions*. v. 3.3.2. January 1991.

[ETS94] *Digital Broadcasting Systems for Television, Sound and Data Services: Framing Structure, Channel Coding and Modulation for 11/12 GHz Satellite Services*. ETS 300 421, DE/JTC-DVB-6, December 1994.

[EUR86] *Cryptanalysts Representation of Non-Linearly Filteres ML-Sequences*. Advances in Cryptology, Eurocrypt'85, Springer-Verlag, pp. 103-110, 1986.

[FUS91] FUSTER, A.; DE LA GUIA, D.; NEGRILLO, J.; MONTOYA, F. *Diseño e implementación de Algoritmos de Generación de Secuencias Binarias*. I Reunión Española de Criptología, Mallorca, 1991.

[GAV96] GAVILÁN, J. J, CRUSELLES, E.; LOPEZ, M. *Transmisión Digital de Televisión DVB.* Mundo Electrónico, Nº 246, Mayo 1996.

[GEF73] GEFFE. *How to Protect Data with Ciphers thar are really hard to break*. Electronics, Jan. 4, 1973.

[GIT92] GITLIN, R. D.; HAYES, J. F.; WEINSTEIN, S. B. *Data Communications Priciples*. Plenum Press, New York, 1992. ISBN 0-306-43777-5.

[GOLIC] GOLIC, J. D.; MIHALJEVIC, M. J. *A Generalized Correlation Attack on a Class of Stream Ciphers Based on the Levenshtein Distance*. Journal of Cryptology, 1991.

[GOL67] GOLD, R. *Optimal Binary Sequences for Spread Spectrum Multiplexing*. IEEE Trans, 1967. IT-13 (5), pp. 619-621.

[GOL68] GOLD,R. *Maximal Recursive Sequences with 3-valued Recursive Coss-correlation Functions*. IEEE Trans., 1968. IT-14 pp.154.

[GOL91] GOLIC, D. J. *A correlation Attack on Ciphers Based in Levehenstain Distance*. Journal of Cryptography, No 3, 1991.

[GRAYB] GRAYBEAL, W. J.; POOCH, U. W. *Simulation: Principles and Methods*. Winthrop Publishers, Inc. Cambridge, Massachusetts.

[HEL84] HELLMAN, M. E.; KARNIN, E. D.; REYNERI, J. *On the Necessity of Cryptanalytic Exhaustive Search*. ACM SIGACT, Vol. 18, Nº 2, pp. 40-44, 1984.

[ISO94] *Coding of Moving Pictures and Associated Audio*. ISO/IEC DIS 13818-1, June 1994.

[JEN80] JENNINGS, S. M. *A Special Class of Binary Sequences*. Ph. D. Thesis. Westfield College, London University, 1980.

[MAS75] MASSEY, J. L.; UHRAN, J. J. *Sub Baud Coding*. Proc. 13th Annual Allerton Conf. on circuit and Systems theory, Monticello III, 1975.

[MAS85] MASSEY, J. L.; INGEMARSSON, I. *The Rip van Winkle Cipher-A Secure and Provably Computationally Secure cipher with Finite Key*.IEEE Int. Symp. on Information theory, Brighton, England, June 24-28, 1985.

[MEI89] MEIER; STAFFELBACH. *Fast Correlation Attacks on Certain Stream Ciphers*. Jourlas of Cryptology, Vol. 1, No 3, 1989.

[MEI92] MEIER; STAFFELBACH. *Correlation Properties of Combiners with Memory in Stream Ciphers*. J. of Cryptology, Vol. 5, No 1, 1992.

[MOU93] MOULY, M.; PAUTET, M. *The GSM System for Mobile Communications*. Europe Media Duplication, S.A., Lassy-Les-Chateaux, 1993.

[PIC82] PICKHOLTZ, R. L.; SCHILLING, D. L.; MILSTEIN, L. B. *Theory of Sprea-Spectrum Communications-A tutorial*. IEEE Transactions on Comm, Vol. COM-30, Nº. 5, May 1982.

[PIP82] PIPER. *Stream Cipher*.Proc. Workshop on Cryptógraphy, Springer-Verlag Lecture Notes in Computer Science, No 149, New York 1982.

[PLE77] PLESS, V. *Encrypting Schemes for Computer Confidentiality*. IEEE Trans on Computer, Vol C-26, No 11,. 1977.

[PUI92] PUIG, M. *Estudio Comparativo de Cifradores en Flujo. Aplicaciones en Comunicaciones Móviles*. Proyecto fin de Carrera. UPC. 1992.

[PUR74] PURSLEY, M. B. *Evaluating Performance of Codes for Spread Spectrum Multiple Access Communications*. 12th Annual Allerton conference on Circuit and Systems Theory, Monticello III, October, 1974.

[PUR77] PURSLEY, M. B.; SARWATE, D. V. *Evaluation of Correlation Parameters for Periodic Sequences*. IEEE Trans, 1977, IT-23 (4), pp.508.

[REC94] RECACHA, F.; BARBA, A.; CRUSELLES, E.; MELUS, J. L. *Comunicaciones Móviles. Seguridad en Redes GSM*. Mundo Electrónico, Nº 251, Octubre 1994.

[RET84] RETTER, C. T. *Cryptanalysis od a MacLaren-Marsaglia System*. Cryptologia, Vol. 8, pp. 97-108, 1984

[RUB79] RUBIN, F. *Decrypting a Stream Cipher Based of J-K Flip-flops*. IEEE Transactions on Computers, Vol. C-28, Nº 7, July 1979.

[RUE84] RUEPPLE, R. A. *New Approaches to Stream Ciphers*. Ph.D.Tesis, Swiss Federal Institute of Technology, zurich, 1984.

[RUE86] RUEPPLE, R. A. *Analysis and Design of Stream Ciphers*. Springer-Verlag. New York 1986.

[RUE89] RUEPPLE, R. A. *Good Stream Ciphers are Hard to Design*. Proc. of 1989 Int. Carnahan conf. on Security Technology. 1989.

[RUEPL] RUEPPLE, R. A. *Design Philosofies for Stream Ciphers*. Proceedings of the Workshop on stream Cciphers. Report 89/1, 1989.

[SARWA] SARWATE; PUSHLEY. *Correlation Properties of Pseudo Random and Related Sequences*.

[SHA49] SHANNON, R.E. *Communication Theory of Secrecy Systems*. Bell Systems Tech. J., Vol 28, No 4, Oct 1949.

[SHA75] SHANNON, R. E. *Simulation: A survey with Research Suggestions*. AIIE Transactions 7, Nº 3, pp. 289-301, Sept. 1975.

[SHA81] SHAMIR, A. *On the Generation of Cryptographically Strong Pseudo-Random Sequences.* 8th Int. colloquium on Automata, Languajes and Programming, Lecture Notes in Computer Science 62, Springer Verlag, 1981.

[SID71] SIDELNIKOV, V. M.. *On Mutual Correlation of Sequences.* Soviet Math Dolk, 1971, 12 (1), pp. 197-201.

[SIE85] SIEGENTHALER, T. *Design of Combiners to Prevent Divide and Conquer Attacks*. Proc. of Crypto'85. Santa Barbara, august, 1985.

[SIEGE] SIEGENTHALER, T. *Correlation Inmunity of Nonlinear Combining Functiones for Cryptograpic Applications*.

[SIEGE] SIEGENTHALER, T. *Decrypting a Class of Stream Ciphers Using Ciphertext Only*. IEEE Trans on Computer, Vol. C-34, No 1, Jan 1985.

[SKAUG] SKAUG, R.; HJELMSTAD, J. F. *Spread Spectrum In Communications*. IEE Telecommuniations Series, v. 12. ISBN 0-86341-034-0.

[SOR96] SORIANO, M. *Contribución al Diseño y Evaluación de Cifradores en Flujo para Comunicaciones Seguras.* Tesis Doctoral. UPC. Barcelona. 1996.

[STA89] STAFFELBACH, O. *Correlation Attacks on Stream Ciphers.* Proccedings of the Workshop on stream Ciphers. Report 89/1, 1989.

[VER26] VERNAN, G. *Cipher Printing Telegraph System for Secret User and Radio Telegraphic Communications*. Journal American Institute of Electrical Engineers XLV, 1926.

[VIT94] VITERBI, A. J. *The Orthogonal-Random Waveform Dichotomy for Digital Mobile Personal Communications*. IEEE Personal Communications, Fisrt Quarter, 1994.

[ZEN89] ZENG, K. C.; YANG, C. H.; RAO, T. R. N. *On the Linear Consistency Tes (LCT) in Cryptanalisis with Applications*. Proc. Crypto'89, Springer-Verlag Lecture Notes in Computer Science, Nº 435, pp.164-174, 1989.

[ZEN88] ZENG, K. C.; HUANG, M. Q. *On the Linear Syndrome Meythod in Cryptanalysis*. Proc. Crypto'88, Springer-Verlag Lecture Notes in Computer Science, Nº 403, pp.469-478, 1988.

[ZEN90] ZENG, K. C.; YANG, C. H.; RAO, T. R. N. *An Improved Linear Syndrome Algorithm in Cryptanalysis with Applications*. Proc. Crypto'90. Springer-Verlag Lecture Notes in Computer Science, 1990.

[ZEN91] ZENG, K. Z.; YANG, C. H.; WEI, D. Y.; RAO, T. R. N. *Pseudorandom Bit Generators in Strem-Cipher Cryptography*. IEEE, 1991.

[ZENGK] ZENG, K.; DAI, Z. D. *On the Linear Predictability Test for Cipher Sequences*

8 Proceso de diseño de un generador

8.0 Introducción

En este capítulo se va a ver un ejemplo completo de diseño y análisis de un generador de secuencias pseudoaleatorias. Para ello, se ha elegido un generador propuesto por E. Cruselles, M. Soriano y J.L. Melús [CRU95]. Tal como se ha visto en capítulos anteriores, siempre podemos especificar ciertas propiedades mínimas que deseamos que cumplan las secuencias que vamos a generar, en función de las aplicaciones en las que vayan a ser utilizadas dichas secuencias. En este caso, se van a definir los siguientes requisitos que deben cumplir las secuencias:

1) Se deben generar familias formadas por un elevado número de secuencias.
2) Todas las secuencias generadas deben mostrar elevado periodo.
3) Las secuencias deben estar incorreladas entre sí.
4) Las secuencias deben mostrar una función de autocorrelación impulsiva.
5) Las secuencias deben tener alta complejidad lineal.
6) Finalmente, deben mostrar buenas propiedades estadísticas.

Estas condiciones son muy restrictivas, por lo que si somos capaces de diseñar un generador cuyas secuencias de salida las cumplan, se podrá utilizar en cualquier aplicación que requiera usar este tipo de secuencias.

Una consideración importante a tener en cuenta en el diseño de un generador es que es un proceso en gran medida heurístico, por lo que no podremos determinar a priori cuáles van a ser las propiedades de la estructura que se haya podido ocurrir. Por tanto, lo más práctico será generar las secuencias y recurrir a los tests que se vieron en el capítulo 5 para comprobar sus propiedades. Con ello, se dispone de una herramienta de diseño importante (que se incluye en un programa con el libro), y que será de gran ayuda a la hora de diseñar estos generadores.

8.1 Diseño de un generador de secuencias pseudoaleatorias

Inicialmente se propondrá un generador que sea lo más elemental posible y que sea capaz de generar secuencias con buenas propiedades de aleatoriedad, además de presentar elevada complejidad a la salida (al que en adelante se denominará PRBS-1). Una vez se disponga de este generador elemental, podrá utilizarse para obtener estructuras más grandes mediante combinaciones de varios de ellos, que

permitan obtener familias de secuencias grandes con gran periodo y elevada complejidad. Para el diseño de este generador elemental se partirá de un registro de desplazamiento con realimentación lineal (LFSR), al que se le introducirá de alguna manera una no linealidad para aumentar la complejidad lineal de las secuencias que produce, puesto que ya se conoce el problema que presenta el LFSR en cuanto a su gran predictibilidad (baja complejidad).

La elección de una función no lineal es siempre delicada, tal como se vio en el capítulo 3, ya que a pesar de aumentar la complejidad de las secuencias que produce el LFSR, generalmente suele estropear la propiedad de máximo periodo y desbalancear los ceros y unos de las secuencias producidas por éste. Una posible solución podría surgir de intentar buscar funciones no lineales cuya salida estuviera balanceada en cuanto a ceros y unos.

Sin embargo, para este generador se ha intentado buscar otra posible solución a este problema, y se ha encontrado ésta en la forma en que se combina la función no lineal con el LFSR. Si se observa la figura 8.1, se puede ver que se ha elegido una estructura en la que el resultado de la función no lineal aplicada sobre las celdas del LFSR, se combina con el resultado de la función de realimentación lineal de éste (mediante la suma módulo 2) para producir el siguiente bit de entrada al LFSR. Aquí está la clave de las buenas propiedades que va a mostrar esta estructura, ya que la función binaria OR-exclusiva de 2 entradas sí tiene una salida balanceada, con lo cual se garantiza una entrada no lineal balanceada al LFSR a pesar de que la salida de la función no lineal no lo esté. Esto hace que la elección de dicha función no lineal no sea tan crítica, aunque después de realizar diversas pruebas los autores llegan a la conclusión de que una AND de tres entradas con una de ellas negada da excelentes resultados para esta estructura, tal como veremos a continuación. Vamos a definir formalmente esta estructura:

8.2 Estructura del generador PRBS-1

Llamemos a_i a los coeficientes del polinomio de realimentación del LFSR, y r_i al contenido de sus celdas $a_i, r_i \in GF(2)$, si $r_i(t)$ es el contenido de la celda r_i después del pulso *t*-ésimo. Entonces, se puede expresar la función de realimentación lineal como $f(t) = a_0 r_0(t) \oplus a_1 r_1(t) \oplus ..... \oplus a_{L-1} r_{L-1}(t)$. Si se asume que $a_0 = 1$, entonces $r_{L-1}(t+1)$ dependerá de $r_0(t)$, ya que de no ser así, no se aprovecharía toda la longitud del LFSR elegido. El valor $a_i = 1$ indica que la celda *i* interviene en la función lineal de realimentación f(*t*) (es decir, que hay conexión), y $a_i = 0$ indica que no hay conexión, es decir, que dicha celda no contribuye al polinomio de realimentación lineal.

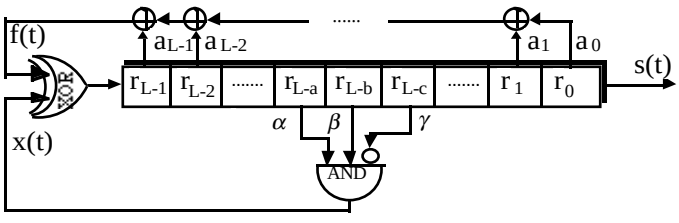


Fig. 8.1 Estructura del generador elemental (PRBS-1)

Tal como se vio en el capítulo 3, un registro de desplazamiento de L celdas $(r_0(t), r_1(t), \ldots, r_{L-1}(t))$ (con t = 0, 1, 2,...) cumple que $r_i(t+1) = r_{i+1}(t)$ para i = 0...L-2, y $r_{L-1}(t+1) = f(r_0(t), r_1(t), \ldots, r_{L-1}(t))$. Si se denota $r_t = r_0(t)$, la secuencia binaria de salida será $\{r_1\} = r_0 r_1 r_2 \ldots$, y para todo registro de desplazamiento realimentado linealmente se cumplirá que:

(1) $r(t)$ estará completamente determinado por $a_0 a_1 \ldots a_{L-1}, r_0 r_1 \ldots r_{L-1}$.

(2) $r_{L-1}(t+1) = \sum_{i=0}^{L-1} a_i r_i(t)$ $\quad t$ = 0, 1, 2, ...

(3) $\{r_1\}$ tendrá periodo $\leq 2^L - 1$.

En el capítulo 3 se vio que un LFSR viene determinado por su polinomio característico $f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{L-1} x^{L-1} + x^L$, donde $a_0 = 1$ y $a_L = 1$. Si f(x) tiene grado L y es primitivo, el periodo de cualquier secuencia no nula de salida será 2^L-1. Esto implica que el registro pasará a través de 2^L-1 estados no nulos antes de repetirse.

Vamos ahora a introducir la no linealidad al LFSR. Si se expresa en la forma algebraica normal (ANF) en GF(2) la función no lineal $x(t)$ elegida para el PRBS-1 de la figura 8.1, las ecuaciones que van a representar a dicho generador serán:

$$r_i(t+1) = r_{i+1}(t) \qquad i = 0, \ldots\ldots, L\text{-}2$$

$$r_{L-1}(t+1) = \left(\sum_{i=0}^{L-1} a_i r_i(t) \right) \oplus \left[\left(r_{L-a}(t) \cdot r_{L-b}(t) \right) \oplus \left(r_{L-a}(t) \cdot r_{L-b}(t) \cdot r_{L-c}(t) \right) \right] \qquad (1)$$

donde $\cdot$ representa la multiplicación binaria (función AND) y $\oplus$ la suma módulo 2 (or-exclusiva). La salida del generador será $s(t) = r_0(t)$, t = 0, 1, 2,... Esto implica que la función no lineal contiene un producto de tres celdas y otro producto de dos celdas, con lo que se tiene una función no lineal de tercer orden.

Para poder realizar una implementación sencilla de dicha función no lineal, y teniendo en cuenta que para una variable binaria se cumple que $(1 \oplus x) = \bar{x}$, parece interesante utilizar una representación no canónica de dicha función que permitirá expresar la parte no lineal de la ecuación (1) como:

$$\left(r_{L-a}(t) \cdot r_{L-b}(t) \cdot \bar{r}_{L-c}(t) \right)$$

Si se impone la restricción de que el polinomio de realimentación lineal del LFSR sea primitivo, y se eligen las conexiones a, b, y c de la función no lineal de forma adecuada (tal como se verá en el apartado 8.3.1), este generador permitirá generar una familia de 2^L-1 secuencias, todas ellas con máximo periodo y máxima complejidad lineal:

$$P_{max} = 2^L - 1 \qquad \Lambda_{max} = 2^L - 2$$

Dada una secuencia producida por el generador, el resto de secuencias de la familia se obtendrá mediante sucesivos desplazamientos de esta secuencia.

Es decir, si $s_1(t)$ es la primera secuencia generada, las otras secuencias serán:

$$s_2(t) = s_1(t-1)$$
$$s_3(t) = s_1(t-2)$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$s_{2^L-1}(t) = s_1(t-2^L+2)$$

Parece que la elección de la estructura ha sido afortunada, ya que se ha conseguido aumentar la complejidad pero sin estropear el máximo periodo que mostraba la secuencia producida por un LFSR. Sin embargo, todavía no se puede tener absoluta certeza sobre ello, puesto que se debe comprobar, además, que las buenas propiedades estadísticas que presenta el LFSR por sí solo todavía se cumplen, recurriendo para ello a los tests estadísticos.

Se estudiarán las propiedades de este generador usando una simulación por ordenador y aplicando a las secuencias obtenidas todos los tests que se explicaron en el capítulo 5, y cuyos programas se incluyen con este libro.

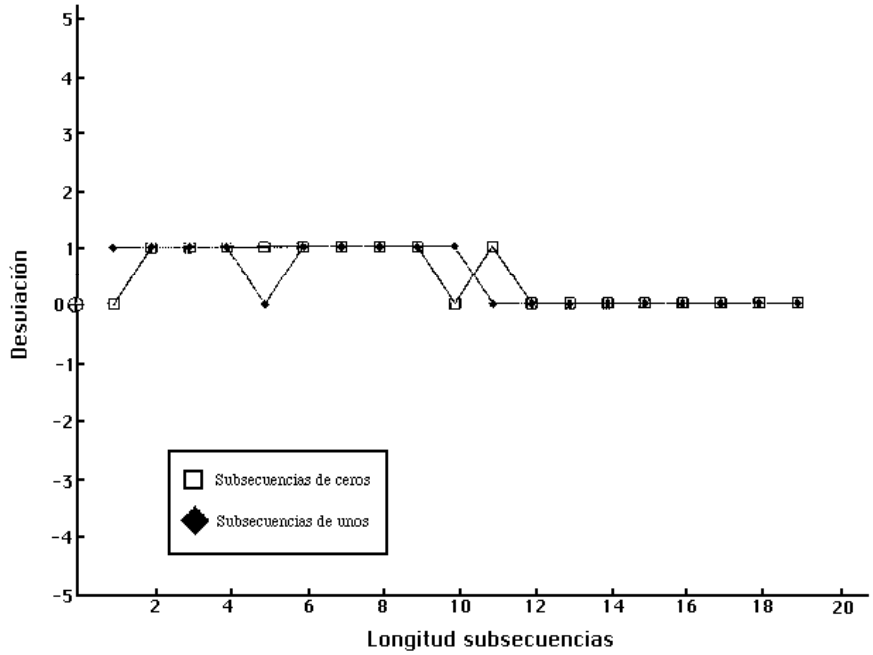
Para ello elegimos un generador constituído por un LFSR de L = 11 celdas, polinomio primitivo de realimentación f(t) = $t^{11}+t^{10}+t^9+t^8+t^3+t+1$ y conexiones de la función no lineal a = 3, b = 4 y c = 10 ($\alpha(t) = r_8(t)$, $\beta(t) = r_7(t)$ y $\gamma(t) = r_1(t)$). Con estos parámetros, se obtiene una familia de 2^{11}-1 = 2.047 secuencias, todas ellas con periodo $P = 2^{11}$-1 = 2.047, y complejidad lineal igual o casi igual a $\Lambda(s(t)) = 2^{11}$-2 = 2.046.

Los resultados obtenidos al estudiar este generador revelan que se comporta muy bien respecto a los tres postulados de Golomb. La columna PRBS-1 de la tabla 8.1, muestra los resultados obtenidos para una de las 2.047 secuencias producidas por este generador. Con respecto al primer postulado, se puede ver que en un periodo de 2.047 bits hay 1.023 ceros y 1.024 unos. Para el segundo postulado, los tests muestran que en un periodo de la secuencia hay 511 subsecuencias, tanto para los ceros como para los unos, y la desviación respecto a la segunda condición de este segundo postulado es muy pequeña, tal como muestra la gráfica 8.1.

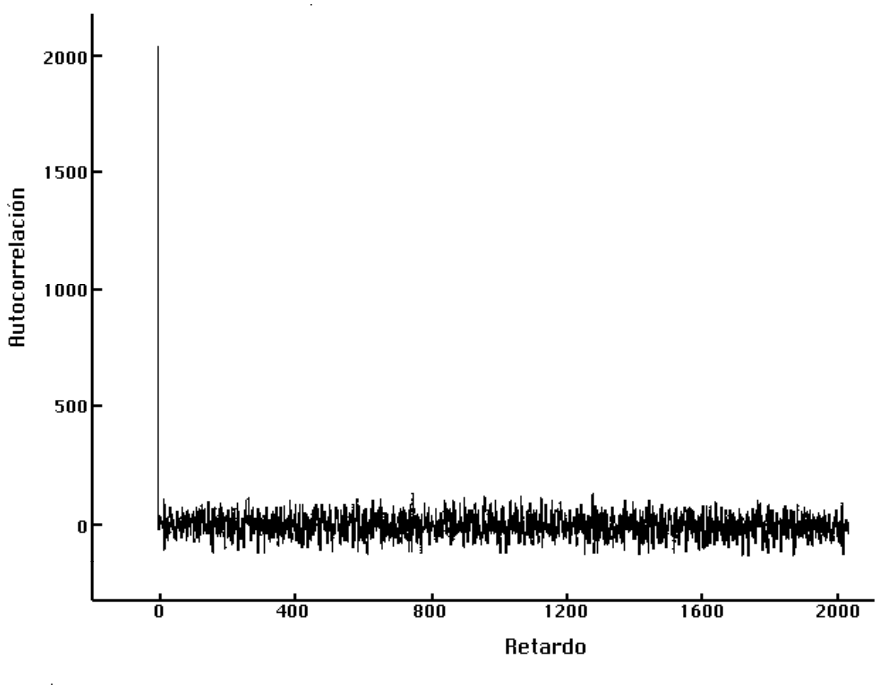
La gráfica 8.2 muestra el comportamiento de la función de autocorrelación normalizada de las secuencias obtenidas, donde se puede ver que tiene un comportamiento aproximadamente impulsivo, con valor unidad en el origen y un valor muy pequeño para los valores fuera de fase. Este comportamiento está muy cercano al que especifica el tercer postulado de Golomb para secuencias aleatorias. Los resultados de esta función también se pueden ver en la fila AC-PRBS-1 de la tabla 8.2. En este caso se muestran los valores sin normalizar, es decir, no están divididos por el periodo de la secuencia.

En el capítulo anterior, se vio que en ciertas aplicaciones como el acceso múltiple por división de código (CDMA) la correlación cruzada entre las secuencias de la familia era de gran importancia, ya que determinaba la interferencia mutua entre los diferentes usuarios que compartían el canal y que estaban asociados a dichas secuencias (o códigos). En la gráfica 8.3 se muestra la correlación cruzada normalizada entre dos secuencias de la familia, donde se puede ver el elevado grado de incorrelación que presentan. Además, el pico máximo obtenido en dicha gráfica y su frecuencia se puede ver en la fila CC-PRBS-1 de la tabla 8.2.

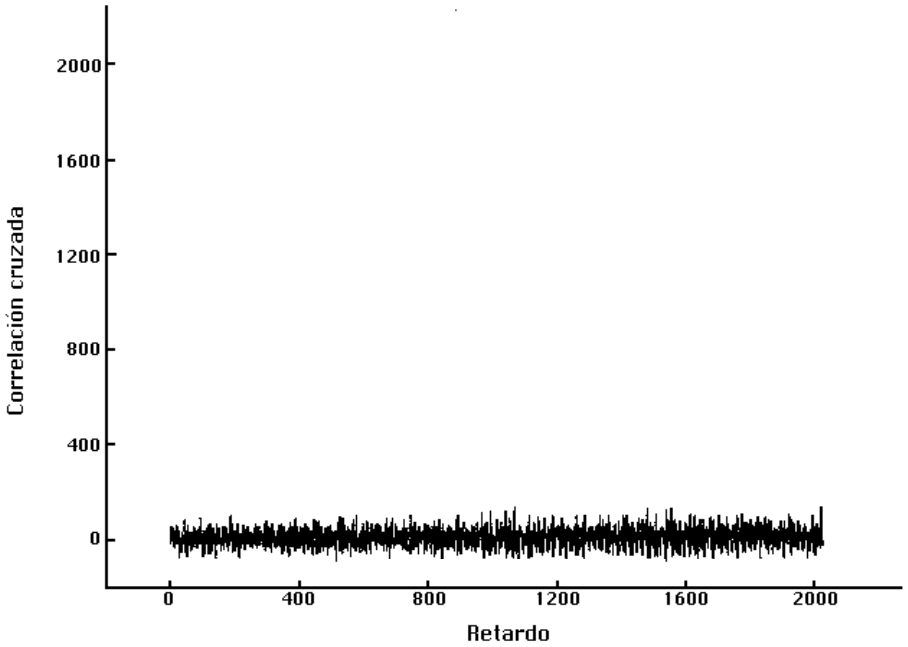
Como ya sabemos, el hecho de añadir una función no lineal al LFSR tiene como objetivo principal aumentar la complejidad lineal de las secuencias producidas por este, disminuyendo de esta



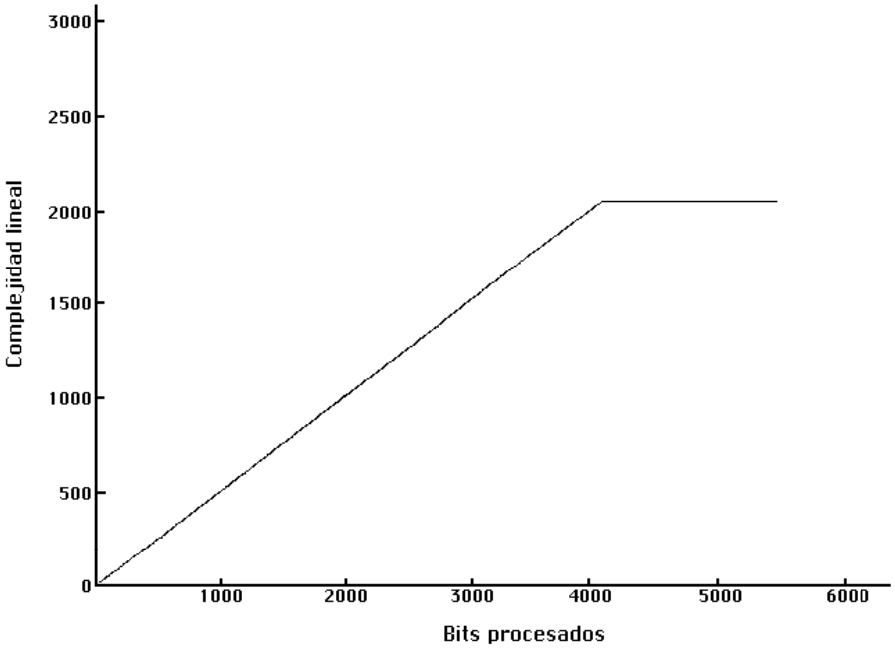
Gráfica 8.1 Desviación respecto al segundo postulado de Golomb para el PRBS-1



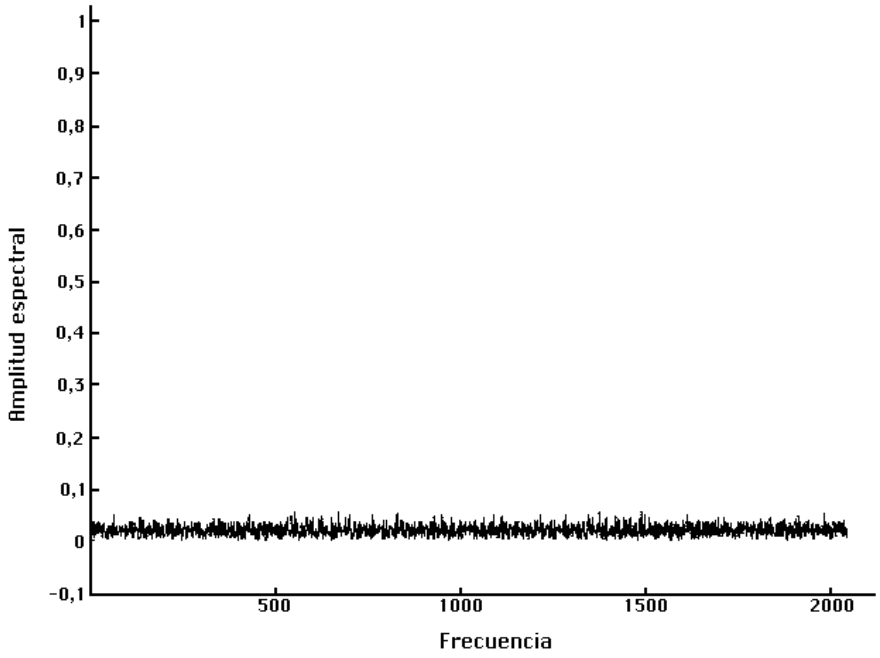
Gráfica 8.2 Función de autocorrelación de PRBS-1



Gráfica 8.3 Correlación cruzada entre dos secuencias producidas por PRBS-1



Gráfica 8.4 Perfil de complejidad lineal (LCP) para el PRBS-1



Gráfica 8.5 Test espectral para PRBS-1

forma su predictibilidad. Este es un requisito esencial que deben cumplir las secuencias si se quiere hacer uso de ellas en aplicaciones como el cifrado en flujo, tal como se vio en el capítulo anterior. Con este generador, obtenemos una complejidad lineal igual, o casi igual, al periodo menos uno $\Lambda(s(t))$ = 2.046, para todas las secuencias de la familia. Dicha complejidad lineal la podemos obtener usando el algoritmo de síntesis de Massey-Berlekemp que se describió en el capítulo 5.

Sin embargo, ya se vio al estudiar los cifradores en flujo que para poder utilizar las secuencias pseudoaleatorias en esta aplicación no sólo debían tener una elevada complejidad lineal, sino que ésta debía crecer, aunque de forma irregular, siguiendo la linea de *k*/2 (donde *k* son los bits de la secuencia procesados mediante el algoritmo anterior).

La gráfica 8.4 muestra que las secuencias producidas presentan un excelente comportamiento respecto al perfil de complejidad lineal (LCP), puesto que la complejidad lineal crece siguiendo esta línea hasta alcanzar el valor de 2.046, después de haber procesado dos periodos de la secuencia utilizando el algoritmo de Massey-Berlekamp.

Un paso esencial para garantizar la bondad del generador diseñado es comprobar que la no linealidad introducida no ha estropeado las buenas propiedades estadísticas que muestra el LFSR. En la columna PRBS-1 de la tabla 8.3 se pueden ver los resultados de los tests estadísticos aplicados sobre una de las secuencias producidas por este generador, y muestran un comportamiento muy bueno en cuanto a la estadística de las secuencias de salida de dicho generador. El valor de 0,9451 obtenido además en la entropía por bit, muy cercano al valor ideal 1, refuerza la conclusión de que las secuencias producidas presentan un excelente comportamiento de aleatoriedad.

En la gráfica 8.5 se muestra el resultado del test espectral, que tal como se vio en el capítulo 5 es un test estadístico teórico basado en la transformada discreta de Fourier de la secuencia (DFT), y muestra también un comportamiento muy cercano al impulsivo, que es el ideal esperado para secuencias aleatorias.

A raíz de todos los resultados vistos en los tests a los que se han sometido las secuencias producidas por este generador, se puede concluir que presentan unas propiedades muy buenas de aleatoriedad, ya que el resultado de los tests es muy parecido al que se produciría con secuencias verdaderamente aleatorias.

Recordemos que uno de los objetivos que se planteaba inicialmente era obtener familias con gran número de secuencias todas ellas de elevado periodo, alta complejidad y buenas propiedades estadísticas.

Si bien estos objetivos ya se pueden cumplir con este generador, puesto que se puede hacer que las dimensiones del LFSR crezcan tanto como se quiera, es más conveniente realizar este crecimiento combinando varios de estos generadores. Por supuesto, deberemos buscar una forma adecuada de hacerlo para no perder las buenas propiedades que ya se hayan conseguido de él.

8.3 Generador basado en combinar varios generadores PRBS-1

En [CRU95], se propone una forma efectiva de combinar *N* generadores PRBS-1 como los descritos en el partado anterior y conseguir así grandes familias de secuencias pseudoaleatorias altamente incorreladas y de elevado periodo, alta complejidad y buenas propiedades estadísticas. A continuación se describe el método empleado:

Se considera un conjunto de *N* generadores PRBS-1 como el de la figura 8.1 y llamamos a_{ij} a los coeficientes del polinomio de realimentación lineal, y r_{ij} a los contenidos de las celdas de los registros de los *N* generadores PRBS-1 ($1 \le i \le L$ y $1 \le j \le N$) con ($a_{ij}, r_{ij} \in GF(2)$). $r_{ij}(t)$ será el contenido de la celda r_j del LFSR-1$_{(i)}$ después del pulso *t*-ésimo.

Denotamos la función de realimentción lineal del LFSR-1$_{(i)}$ como $f_i(t) = a_{i_0} r_{i_0}(t) \oplus a_{i_1} r_{i_1}(t) \oplus \cdots \oplus a_{i_{L-1}} r_{i_{L-1}}(t)$. Asumimos que $a_{i_0} = 1$, y que $r_{i_{L-1}}(t)$ depende de $r_{i_0}(t)$, ya que de otra forma no se aprovecharía toda la longitud de los LFSRs. Finalmente consideramos que $a_{ij} = 1$ denota una conexión cerrada, y $a_{ij} = 0$ una conexión abierta, es decir, que en este último caso dicha celda no contribuye al polinomio lineal de realimentación del LFSR-1 correspondiente.

El generador obtenido, que se muestra en la figura 8.2 (y que llamaremos PRBS-2), combina *N* generadores PRBS-1 (donde L_1, L_2,..., L_N son el número de celdas de éstos respectivamente) de la siguiente forma: el resultado de la función no lineal de cada generador PRBS-1 ($x_i(t)$ para $1 \le i \le N$-1) se suma, módulo 2, con el resultado del polinomio de realimentación lineal del siguiente generador y el resultado de la función no lineal del último generador PRBS-1 ($x_N(t)$) se usa para generar el siguiente bit del primero (después de haberse sumado, módulo 2, con el resultado de su función lineal de realimentación).

Esta estructura permite generar *M* secuencias pseudoaleatorias donde:

$$M = N\left(2^{\sum_{i=1}^{N} L_i} - 1\right)$$

La generación de estas *M* secuencias es la siguiente: cada secuencia obtenida como un desplazamiento de cada una de las *N* secuencias anteriores ($s_1(t-\tau)$, donde $1 \le t \le N$ y $0 \le \tau \le 2^{\sum_{i=1}^{N} L_i} - 1$) se podrá considerar como un nuevo elemento de la familia y tendrá también las propiedades que se verán más adelante.

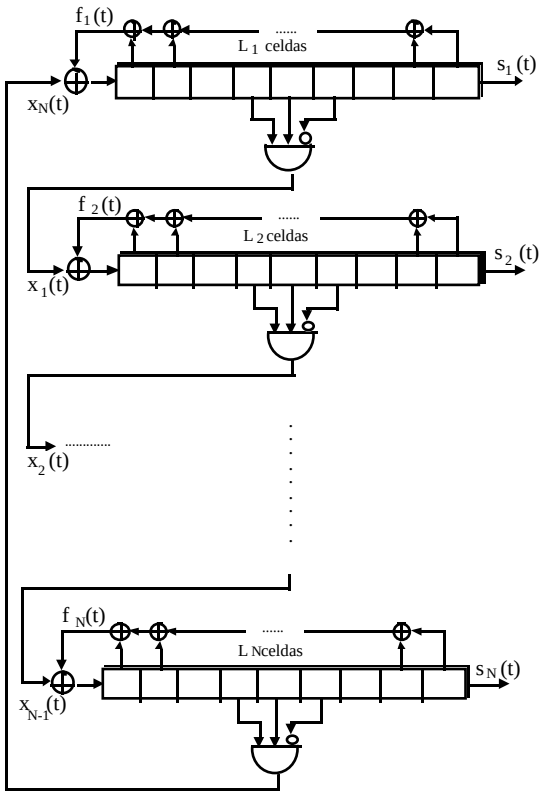


Fig. 8.2 Generador PRBS-2

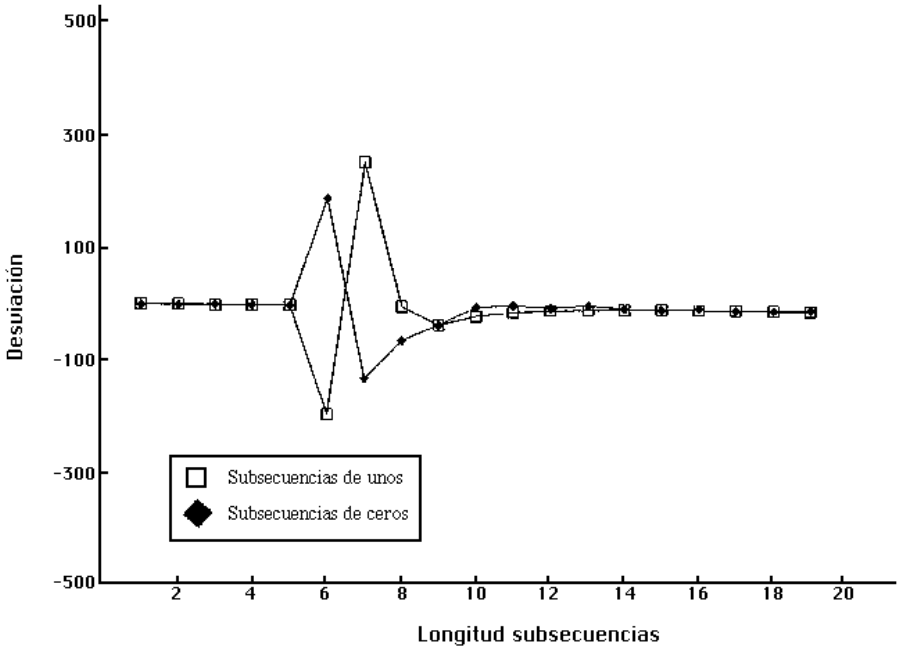
Las ecuaciones que definirán este generador serán:

$$r_{p_i}(t+1) = r_{p_{i+1}}(t) \qquad \text{donde} \quad i = 0,\ldots,L_p - 2 \quad \text{y} \quad 1 \le p \le N$$

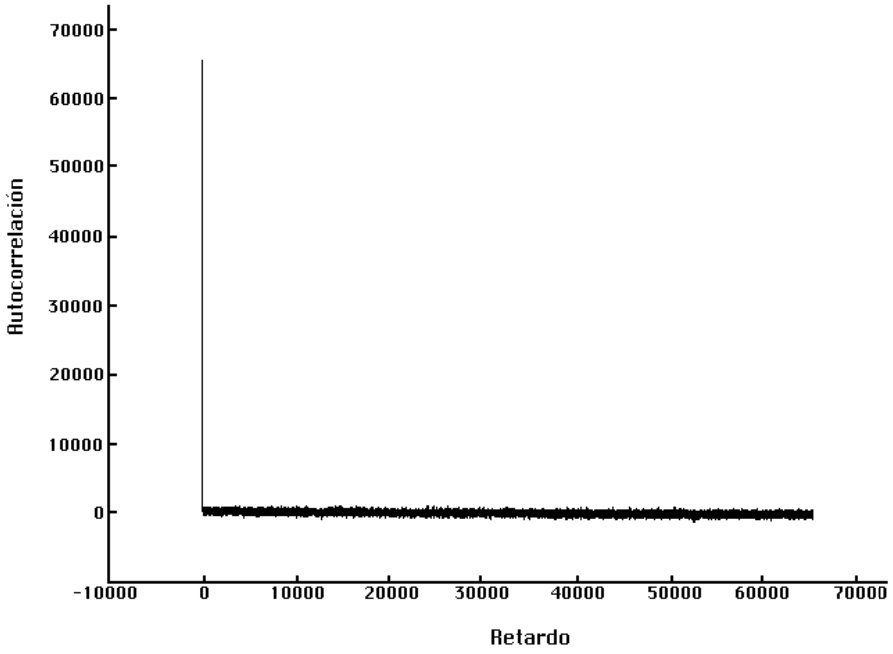
$$r_{L_1-1}(t+1) = \left(\sum_{j=0}^{L_1-1} a_{1_j} r_{1_j}(t) \right) \oplus \left[\left(r_{L_N-a_N}(t) \cdot r_{L_N-b_N}(t) \cdot r_{L_N-c_N}(t) \right) \oplus \left(r_{L_N-a_N}(t) \cdot r_{L_N-b_N}(t) \right) \right]$$

$$r_{Lp-1}(t+1) = \left(\sum_{j=0}^{Lp-1} a_{p_j} r_{p_j}(t) \right) \oplus \left[\left(r_{L_{p-1}-a_{p-1}}(t) \cdot r_{L_{p-1}-b_{p-1}}(t) \cdot r_{L_{p-1}-c_{p-1}}(t) \right) \oplus \left(r_{L_{p-1}-a_{p-1}}(t) \cdot r_{L_{p-1}-b_{p-1}}(t) \right) \right]$$

$$\text{donde} \quad 1 < p \le N$$



Gráfica 8.6 Desviación respecto al segundo postulado de Golomb para PRBS-2



Gráfica 8.7 Función de autocorrelación de la secuencia S1 del PRBS-2

Las *M* secuencias producidas por este generador mostrarán periodo máximo y complejidad lineal igual, o casi igual, a este periodo máximo.

Ambas vendrán determinadas por las expresiones:

$$P_{max} = 2^{\sum_{i=1}^{N} L_i} - 1 \qquad \text{y} \qquad \Lambda_{max} \approx P_{max} - 1 = 2^{\sum_{i=1}^{N} L_i} - 2$$

donde L_i es el número de celdas del registro de desplazamiento del generador PRBS-$1_{(i)}$.

Para obtener estos buenos resultados en cuanto a periodo y complejidad se deben elegir cuidadosamente las conexiones (*a*, *b* y *c*) de las funciones no lineales de todos los generadores PRBS-1.

Sin embargo, en este caso, puesto que todos ellos están entrelazados, esta elección se debe fundamentar en obtener las conexiones óptimas que permitan obtener la máxima eficiencia de toda la estructura en su conjunto.

La estructura PRBS-1 del apartado anterior puede verse ahora como un caso especial de la estructura general PRBS-2 cuando $N = 1$.

Al igual que se hizo en el apartado anterior, antes de dar por buena esta estructura se debe someter las secuencias que produce a toda una batería de tests de aleatoriedad.

Para ello se elige una estructura con 3 generadores del tipo PRBS-1 (N=3) con L_1= 5, L_2= 7 y L_3= 4 celdas y polinomios de realimentación lineal primitivos $f_1(t) = t^5+t^2+1$, $f_2(t) = t^7+t+1$ y $f_3(t) = t^4+t+1$ respectivamente. Como conexiones óptimas para las funciones no lineales de los generadores PRBS-1 se eligen:

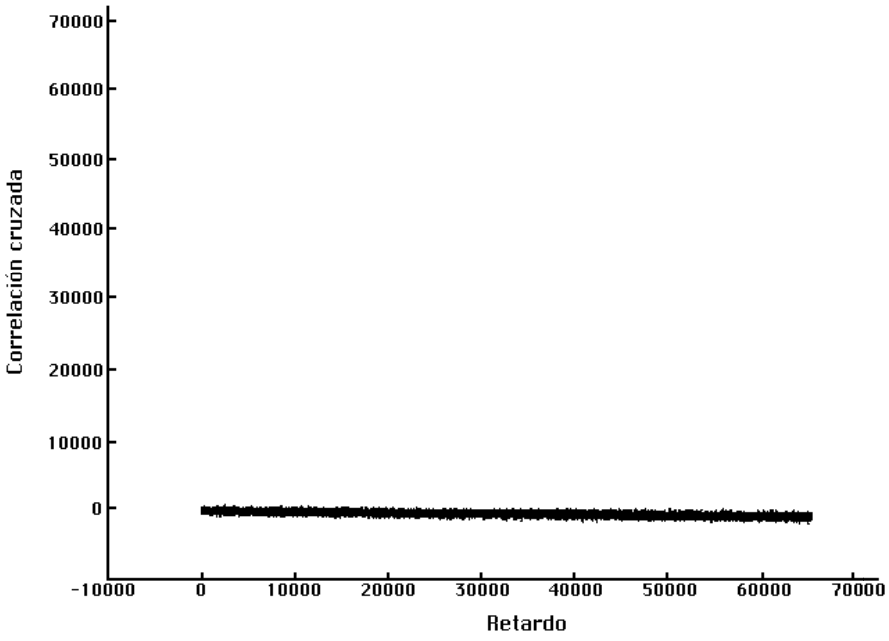
Generador 1: a_1= 4, b_1= 1 y c_1= 2 ($\alpha_1(t) = r_{10}(t)$, $\beta_1(t) = r_{14}(t)$, $\gamma_1(t) = r_{13}(t)$)
Generador 2: a_2= 4, b_2= 3 y c_2= 6 ($\alpha_2(t) = r_{23}(t)$, $\beta_2(t) = r_{24}(t)$, $\gamma_2(t) = r_{21}(t)$)
Generador 3: a_3= 3, b_3= 3 y c_3= 1 ($\alpha_3(t) = r_{31}(t)$, $\beta_3(t) = r_{31}(t)$, $\gamma_3(t) = r_{23}(t)$)

Con esta estructura obtendremos $M = 3\times(2^{16}-1) = 3\times65.535 = 196.605$ secuencias, todas ellas de periodo 65.535 y complejidad lineal igual, o casi igual a 65.534, que son los máximos valores que se pueden obtener teóricamente.

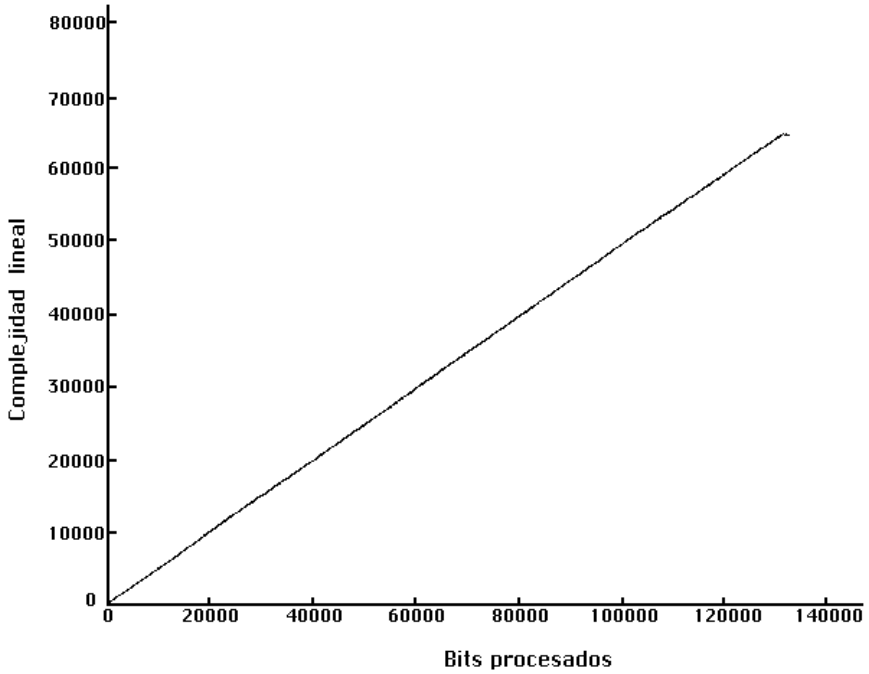
Las tres primeras columnas de la tabla 8.1 muestran estos resultados para tres de las secuencias producidas por el generador ($s_1=s_1(t\text{-}0)$, $s_2=s_2(t\text{-}0)$ y $s_3=s_3(t\text{-}0)$), y que se han llamado *S1*, *S2* y *S3* respectivamente.

Tabla 8.1 Resultados a los tests de periodo, complejidad lineal y 2º postulado de Golomb

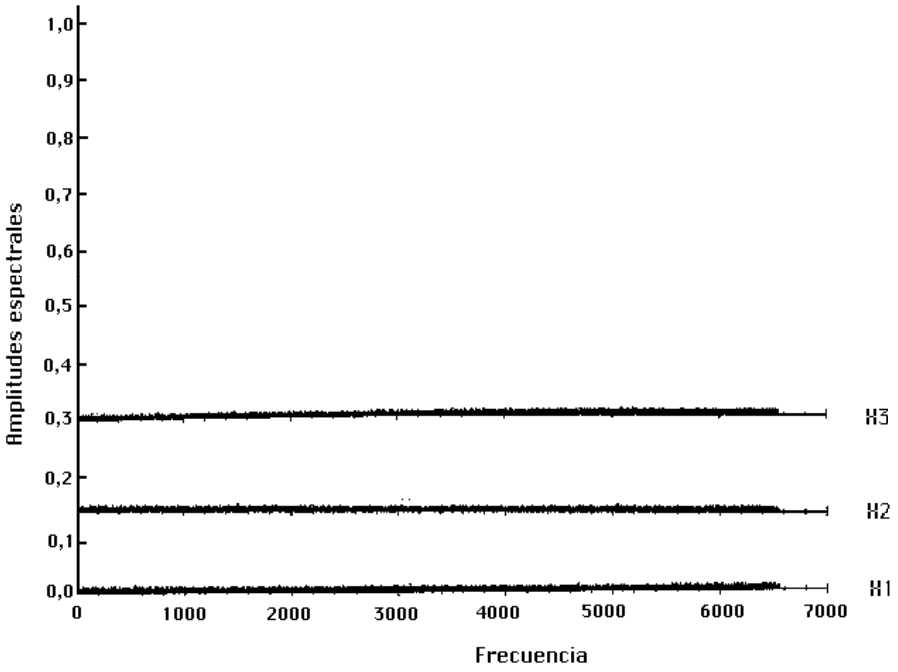
	Sec. S1	*Sec. S2*	*Sec. S3*	*PRBS-1*
Periodo	65.535	65.535	65.535	2.047
Complejidad Lineal	65.518	65.534	65.534	2.046
Nº de unos	32.768	32.768	32.768	1.024
Nº de ceros	32.767	32.767	32.767	1.023
Nº subsecuen. unos	16.383	16.384	16.392	511
Nº subsecuen. ceros	16.383	16.385	16.395	511



Gráfica 8.8 Correlación cruzada entre las secuencias S1 y S2 del PRBS-2



Gráfica 8.9 Perfil de complejidad lineal (LCP) para Si ($1 \leq i \leq 3$) del PRBS-2.



Gráfica 8.10 Resultado del test espectral para Si ($1 \leq i \leq 3$) del PRBS-2

Todas las secuencias producidas por este generador han mostrado muy buen comportamiento con respecto a los tres postulados de Golomb. En la tabla 8.1 se pueden ver los resultados obtenidos para las tres secuencias de la familia estudiadas, donde las filas 3 y 4 corresponden al primer postulado, y las filas 5 y 6 al segundo. Además, la desviación respecto al segundo postulado de Golomb para la secuencia *S1* se muestra en la gráfica 8.6, tanto para ceros como para unos.

Con respecto al tercer postulado de Golomb, la gráfica 8.7 muestra la función de autocorrelación periódica de la secuencia *S1*, donde se puede observar que muestra un comportamiento casi impulsivo, con valor unidad en el origen y sin picos para valores fuera de fase, tal como se espera de las secuencias aleatorias.

Tabla 8.2 AC: Autocorrelación, CC: Correlación cruzada

	Periodo (P)	*Valor Origen (VO)*	*Pico fuera-de-fase max. (MPF)*	*Pico fuera-de-fase min.*	*Nº. de veces MPF aparece (D)*	*Frecuencia (F=D/P)*	*LPLS= MPF/VO*
AC-PRBS1	2.047	2.047	138	2	2	0,09 %	0,067
CC-PRBS1	2.047	-22	130	2	1	0,04 %	-------
AC-PRBS2	65.535	65.535	1.092	0	1	0,001 %	0,016
CC-PRBS2	65.535	-10	1.114	2	1	0,001 %	-------

En la fila AC-PRBS2 de la tabla 8.2 se pueden ver los valores máximos de dicha función y su frecuencia, teniendo en cuenta que en dicha tabla aparecen valores no normalizados. El que sean valores no normalizados implica que son los de la gráfica 8.7 pero multiplicados por el periodo de la secuencia, es decir, por 65.535 (en la función no normalizada el valor del origen de la función de autocorrelación no sería 1 sino precisamente 65.535).

En la gráfica 8.8 se puede ver la función de correlación cruzada periódica entre las secuencias *S1* y *S2* que muestra el elevado grado de incorrelación que presentan. Esto implica que será fácil distinguirlas entre otras secuencias de la familia. Además, en la fila CC-PRBS-2 de la tabla 8.2 se pueden ver los máximos picos obtenidos y su frecuencia (al igual que antes, en la tabla se muestran los valores de la función de correlación cruzada no normalizada).

Una vez analizados los postulados de Golomb y las propiedades de correlación, es necesario estudiar la complejidad de las secuencias obtenidas.

La fila 2 de la tabla 8.1 muestra que la complejidad lineal es igual o casi igual al periodo para todas las secuencias. Sin embargo, ya sabemos que esto no es una condición suficiente para tener elevada aleatoriedad, y también es necesario que el perfil de complejidad lineal (LCP) siga, aunque de forma irregular, la línea de $k/2$, si k es el número de bits procesado. La gráfica 8.9 muestra el buen comportamietno del LCP de las secuencias estudiadas. Si se utilizan estas secuencias en una aplicación de cifrado en flujo, ésta sería una condición suficiente para hacer fracasar los ataques basados en el algoritmo de Massey-Berlekemp.

Además, para que se pueda considerar a las secuencias generadas como aleatorias, éstas deben mostrar un buen comportamiento estadístico.

Para detectar posibles defectos estadísticos, se puede hacer pasar las secuencias por una serie de tests que comprueben una muestra de la secuencia de una cierta longitud y rechacen al generador cuando ciertas propiedades de la muestra sugieran una posible no aleatoriedad (por ejemplo, que el número de ceros y unos difiera sustancialmente). Un defecto estadístico en el generador se podrá detectar con una cierta probabilidad que dependerá de la gravedad del defecto y de la longitud de la muestra estudiada. En la práctica, suele ser suficiente con pasar las secuencias por una batería de una docena de tests , y si consigue pasar la mayoría de ellos, la podemos considerar aleatoria.

La tabla 8.3 muestra los buenos resultados obtenidos en todos los tests estadísticos considerados para las tres secuencias estudiadas. Cada uno los tests (excepto los tests estadísticos basados en el LCP) se ha aplicado 10 veces sobre diversas partes de la secuencia empleando un nivel de significancia del 5%, y sobre sus resultados se ha aplicado un test de Kolmogorov-Smirnov usando la salida de éste como una indicación de si la secuencia pasa o no el test.

Otro test estadístico de aleatoriedad que se puede utilizar para testear la aleatoriedad de las secuencias es el de entropía por bit. A diferencia de los otros tests, que buscan defectos estadísticos concretos sobre las secuencias, este test es capaz de detectar cualquier tipo de defecto estadístico aunque, sin embargo, si lo detecta, no es capaz de decir cuál es. Para una secuencia verdaderamente aleatoria, la entropía por bit será uno o muy cercana a 1.

La tabla 8.3 muestra un valor de entropía por bit muy cercano a uno para las tres secuencias estudiadas, lo que refuerza la conclusión que ya se había obtenido a partir de los otros tests de la buena aleatoriedad de las secuencias producidas.

Finalmente, se pueden estudiar las secuencias mediante el test espectral. Tal como se vio en el capítulo 5, las secuencias de periodo P realmente k-distribuídas mostrarán un resultado a este test con valor 1 a la frecuencia 0 y con valor $1/P$ para las otras frecuencias. La gráfica 8.10 muestra un comportamiento muy cercano al ideal para las tres secuencias estudiadas.

Tabla 8.3 Resultados de los tests estadísticos

Test	*Sec. S1*	*Sec. S2*	*Sec. S3*	*PRBS-1*
Frequencia	SI (45%)	SI (45%)	NO (>100%)	SI (45%)
Parejas	SI (13%)	SI (27%)	SI (68%)	SI (50%)
Tríos	SI (56%)	SI (94%)	SI (24%)	SI (72%)
Cuartetos	SI (93%)	SI (50%)	SI (50%)	SI (11%)
Transiciones	SI (90%)	SI (72%)	SI (50%)	SI (72%)
Corr. Serie	SI (60%)	SI (57%)	SI (94%)	SI (34%)
"Runs up"	SI (60%)	NO (<1%)	NO (<1%)	NO (<1%)
"Runs down"	NO (<1%)	NO (<1%)	NO (<1%)	NO (<1%)
Permutaciones	NO (<1%)	SI (6%)	NO (<1%)	SI (62%)
Colect. Cupones	NO (<1%)	NO (<1%)	NO (<1%)	NO (<1%)
"Runs above"	SI (56%)	SI (13%)	SI (71%)	SI (94%)
"Runs below"	SI (23%)	SI (43%)	SI (34%)	SI (38%)
Saltos LCP	SI	SI	SI	SI
Alturas LCP	SI	SI	SI	SI
Poker	SI (34%)	NO (<1%)	NO (<1%)	SI (17%)
Entropía por bit	0,9880	0,9768	0,9654	0,9451

Con esta estructura, también será posible obtener otros periodos en el rango $1...P_{max}$, seleccionando las conexiones apropiadas en las funciones no lineales de los generadores PRBS-1. En la tabla 8.4 se pueden ver los resultados obtenidos para algunos de estos subgeneradores con periodos subóptimos.

Se puede observar que en algunos casos, se pueden llegar a obtener los mismos resultados con funciones no lineales de segundo orden, ya que $a = b$.

Además, la tabla 8.4 muestra que con estas secuencias subóptimas se obtienen las mismas propiedades que se han visto para el caso de conexiones óptimas con respecto a la complejidad lineal y a los postulados de Golomb.

Ante los resultados obtenidos se puede concluir que se han conseguido los objetivos que se pretendían al inicio del capítulo. Se ha diseñado una estructura capaz de producir grandes familias de secuencias pseudoaleatorias, en las que todos sus elementos presentan elevado periodo y alta complejidad lineal, están muy incorreladas entre sí y presentan, además, buenas propiedades estadísticas.

Además, los tests estadísticos han sido de gran utilidad para poder comprobar las propiedades de las secuencias producidas. Sin embargo, una vez que se sabe que estas secuencias presentan una calidad aceptable como secuencias aleatorias, se puede plantear la posibilidad de realizar un estudio analítico más profundo sobre la estructura diseñada.

Si bien es muy complejo realizar este tipo de estudio en estructuras con realimentación lineal como la que se ha estudiado en este capítulo, en el próximo apartado se van a ofrecer algunas reglas de diseño que conducirán a que las secuencias producidas por la estructura diseñada presenten las propiedades que hemos visto en el estudio anterior.

Tabla 8.4 Resultados para conexiones no lineales no óptimas

Periodo (P)	*CONEXIONES*									*Complejidad Lineal*	*Nº de unos*	*Nº de ceros*	*Nº subsec de unos*	*Nº subsec de ceros*
	a1	*a2*	*a3*	*b1*	*b2*	*b3*	*c1*	*c2*	*c3*					
30	0	1	2	4	5	1	1	2	0	29	15	15	7	7
74	0	1	2	5	3	0	2	2	0	73	37	37	18	18
150	1	1	2	2	2	4	2	2	0	147	75	75	37	37
251	0	1	2	5	4	1	0	2	1	250	126	125	62	62
511	0	1	2	3	4	2	1	2	0	510	256	255	128	128
1.096	0	1	2	0	3	5	1	2	0	1.095	548	548	274	274
2.051	0	1	2	3	3	0	0	0	1	2.051	1.026	1.025	513	513
4.045	0	1	2	3	5	1	2	0	1	4.043	2.023	2.022	1.010	1.010
8.164	0	1	2	0	3	1	1	1	0	8.164	4.082	4.082	2.039	2.039
16.330	0	1	2	0	1	3	0	1	2	16.328	8.165	8.165	4.080	4.080
32.524	0	1	2	0	0	4	2	2	0	32.523	16.262	16.262	8.131	8.131

8.3.1 Reglas de diseño de la estructura del generador

Como ya se ha visto en el apartado anterior, necesitamos ciertas reglas para determinar, dado un periodo, los parámetros del generador PRBS-1 L, *a*, *b* y *c* que hacen que se obtengan los excelentes resultados que han revelado los tests anteriores. El número de posibles combinaciones de las conexiones no lineales *a*, *b* y *c* que hacen que se obtenga el máximo periodo para el generador PRBS-1, una vez fijado *L*, es bastante considerable, tal como muestra la tabla 8.5.

Tabla 8.5 Combinaciones óptimas según el número de celdas del registro de desplazamiento

Nº de celdas	4	5	6	7	8	9	10	11	12
Combinaciones	15	52	116	308	480	648	1.032	1.460	2.166

Para el generador PRBS-1 que se ha propuesto, será necesario que se cumplan las siguientes reglas para obtener el máximo periodo y la máxima complejidad lineal:

R1) Hay que evitar que cualquier conexión no lineal (*a*, *b*, o *c*) coincida con la celda *L*. Esto evitará que aparezca cualquier tipo de preámbulo en las secuencias generadas y permitirá alcanzar el periodo máximo.

R2) No es necesario que el polinomio de realimentación sea primitivo para conseguir máximo periodo, ya que se trata de una realimentación no lineal. Sin embargo, el número de coeficientes $a_i = 1$ del polinomio de realimentación debe ser impar para garantizar que dicho polinomio no sea múltiplo de $x+1$.

R3) Aunque las conexiones no lineales *a* y *b* pueden coincidir, la conexión *c* debe ser diferente de *a* y *b*, ya que si no el generador se comportaría como un LFSR.

Si se cumplen estas reglas básicas, se obtendrán las siguientes propiedades del generador:

P1) Sea *G* un generador PRBS-1 de *L* celdas con polinomio de realimentación f(*x*), conexiones no lineales *a*, *b* y *c*, que genera una secuencia pseudoaleatoria de periodo *P*. Será posible obtener el mismo periodo con otro generador de *L* celdas con polinomio de realimentación $f^*(x)$, conexiones no lineales *a' = L-a*, *b' = L-b* y c' = *L-c*. Si $f(x) = a_L + a_{L-1}x + a_{L-2}x^2 + ... + a_1x^{L-1} + a_0x^L$, entonces $f^*(x) = a_0 + a_1x + + a_2x^2 + ...+ a_{L-1}x^{L-1} + a_Lx^L$.

P2) Sean *G* y *G'* dos generadores PRBS-1 de *L* celdas donde los coeficientes del polinomio de realimentación f(*x*) son $a_i, a_j,...a_k = 1$, y los restantes son 0, y los coeficientes de f'(*x*) son $a_{i'}, a_{j'},....., a_{k'} =1$, y el resto 0. Si *i, j, k = i'+p, j'*+p,..., *k'+p* módulo *L*+1, se obtendrá la misma secuencia si las correspondientes conexiones no lineales *a*, *b*, y *c* son iguales a *a'+p*, *b'+p*, y *c'+p* respectivamente.

P3) Sean *G* y *G'* dos generadores con el mismo número de celdas donde los coeficientes del polinomio de realimentación f(*x*) son $a_i, a_j,..., a_k = 1$, y los restantes son iguales a 0, mientras que los coeficientes de f'(*x*) son $a_{i'}, a_{j'},... a_{k'} = 1$ y el resto iguales a 0. El primer generador con conexiones no lineales *a*, *b* y *c* generará una secuencia pseudoaleatoria de periodo *P*. Si *i, j,..., k* = $m_i, m_j,..., m_k$ módulo *P*, y el m.c.d.(*P*,*m*)=1, se podrá obtener el mismo periodo con las conexiones no lineales *a'*, *b'* y *c'* si *a*, *b* y *c* son *ma'*, *mb'* y *mc'* respectivamente.

8.4 Bibliografía

[CRU95] CRUSELLES, E.; SORIANO, M.; MELUS, J.L. *Uncorrelated PN Sequences Generator for Spreading Codes in CDMA Systems*. 6th IEEE International Sysmposium on Personal, Indoor and Mobile Radio Communications (PIMRC´95). September 27-29, 1995. Toronto, Canada

[CRU96] CRUSELLES, E.; SORIANO, M.; MELUS, J.L. *Spreading Codes Generator for Wireless CDMA Networks*. Wireless Personal Communications. Kluwer Academic Publishers. 1996.

[CRUSE] CRUSELLES, E.; SORIANO, M.; FORNE, J.; MELUS, J.L. *Secure Communication in Broadband Networks*. 3rd. International Conference on Telecommunications Systems. Nashville, Tenesee, March 95.

[PUI92] PUIG, M. *Estudio Comparativo de Cifradores en Flujo. Aplicaciones en Comunicaciones Móviles*. Proyecto fin de Carrera. UPC. 1992.

[PUI92J] PUIG, M.; CRUSELLES, E.; J. L. MELÚS. *A Unified Evaluation of Some Proposed Stream-Ciphers.* The 15th Symposium on Information Theory (SITA`92). September 8-11, 1992. Minakami, Japan.

[PUI92M] PUIG, M.; CRUSELLES, E.; J. L. MELÚS. *Algunas Especificaciones de Diseño de los Cifradores de Flujo.* II Reunión Española sobre Criptología. Octubre 1992. Madrid.

[SOR90] SORIANO, M.; FORNE, J.; CRUSELLES, E.; MELUS, J.L. *Linear Complexity Stability in Stream Ciphering for High Speed Networks*. 4th. International Conference on

Telecommunications Systems. Modelling and Analysis Nashville, Tenesee, March 21-24, 1996.

[SOR96] SORIANO, M. *Contribución al Diseño y Evaluación de Cifradores en Flujo para Comunicaciones Seguras*. Tesis Doctoral. UPC. Barcelona. 1996.

9 PREDICTEST: Programa de análisis de secuencias pseudoaleatorias

9.0 Introducción

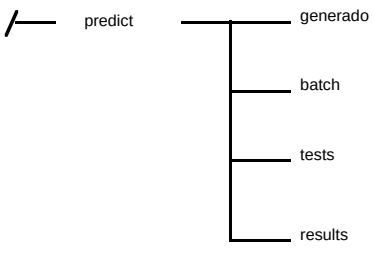
Tal como se ha visto en los capítulos anteriores, al hacer un diseño de un generador de secuencias pseudoaleatorias es conveniente hacer un chequeo de sus propiedades, para comprobar si cumple las especificaciones de aleatoriedad mínimas necesarias para considerarlo útil para la aplicación considerada. Esta comparación se puede llevar a cabo pasando la secuencia de salida del generador de secuencias por una serie de tests que nos permitan evaluar hasta qué punto cumple las especificaciones requeridas.

En este capítulo se describe el funcionamiento del programa PREDICTEST, desarrollado por los autores para realizar este testeo intensivo de las secuencias producidas por los generadores de secuencias pseudoaleatorias. El lenguaje de programación utilizado ha sido el C estándar y, para mayor claridad, se han incluído algunos comentarios en ellos. El código ha sido además optimizado y vectorizado para poder aprovechar las posibilidades de ciertas máquinas con capacidad de proceso vectorial. Este tipo de ordenadores pueden hacerse necesarios cuando las longitudes de las secuencias pseudoaleatorias que se deben estudiar comienzan a tener periodos considerables.

Si bien en este capítulo se verá un ejemplo práctico de utilización de este programa, en el capítulo anterior ya se utilizó para realizar el análisis de las estructuras propuestas.

9.1 Instalación del programa

La estructura de directorios sobre la que se debe basar la instalación sobre una estación de trabajo con sistema operativo UNIX debe ser la siguiente:



En el directorio *generado* se deben colocar los programas de los generadores que se desean testear. El programa contiene algunos programas de generadores en este directorio, algunos de de los cuales, como por ejemplo los LFSR, se pueden aprovechar para otros generadores que el lector quiera programar. En este directorio se pueden encontrar los programas que implementan los siguientes generadores:

- *Caótico.c*: Implementa el generador descrito en el capítulo 4.
- *Jennings.c*: Implementa el generador descrito en el capítulo 4.
- *Gollmann.c*: Implementa el generador descrito en el capítulo 4.
- *Lfsr.c*: Implementa un LFSR como el descrito en el capítulo 3.
- *PRBS-1.c* : Implementa el generador PRBS-1 del capítulo 8.
- *PRBS-2.c*: Implementa el generador PRBS-2 del capítulo 8.

Además, en el directorio *batch* se pueden encontrar los correspondientes ficheros batch para estos programas. El directorio *tests* contiene los programas de los tests implementados, que se pueden ver en la tabla 9.1. Si se desarrolla algún otro test debe colocarse en este directorio. Aquí se debe colocar la secuencia de salida de los generadores bajo estudio (fichero llamado *SECUEN*). También contiene un fichero llamado *parám.h*, que contiene los parámetros necesarios para configurar los tests (como por ejemplo la longitud de la secuencia, el número de veces que se realizarán los tests chi-cuadrado en los tests estadísticos, etc). Puesto que todos los tests buscan este fichero, en el momento de instalación se debe modificar la línea:

```
#include "/predict/tests/param.h"
```

adecuadamente para que puedan encontrar este programa (en el caso en que el lector recurra a otra estructura de directorios). El directorio *batch* incluye el progama principal *predict.bat*, que es un fichero *batch* que contiene las llamadas a todos los tests anteriores para ejecutarlos secuencialmente y que envía los ficheros de resultados obtenidos al directorio *results*. Por tanto hay que modificar adecuadamente la línea:

```
# cd /predict/tests
```

que se encuentra al inicio de este fichero *batch*, de forma que se coloque en el directorio *tests.dir*, que es donde están los programas de tests que se van a ejecutar, y todas las líneas que contengan

```
# mv /predict/tests/fichero_resultado   /predict/results
```

de forma que pasen los ficheros de resultados al directorio *results*.

En caso de que no se quieran pasar todos los tests que incluye este fichero, se puede hacer una nueva versión del fichero *predict.bat* eliminando las llamadas a los tests que no interesen. Por otra parte, si se crea un nuevo test, además de colocar su programa fuente en el directorio *tests*, hay que poner en este fichero *batch* su llamada. En el directorio *results* aparecerán todos los ficheros de resultados una vez que se haya ejecutado correctamente el programa *predict.bat.* Posteriormente se hará una descripción de estos ficheros de resultados y de cómo extraer conclusiones a partir de ellos.

Tabla 9.1 Ficheros que implementan tests en el directorio tests y sus ficheros de resultados

TEST	*DESCRIPCION*	*FICHERO RESULTADO*
t-per.c	Determina el periodo de una secuencia	RESPER
t-ccn.c	Calcula la correlación cruzada normalizada de dos secuencias	RESCCN
t-ccnn.c	Calcula la correlación cruzada no normalizada de dos secuencias	RESCCNN
t-gol1.c	Determina el cumplimiento del primer postulado de Golomb	RESG1
t-gol2.c	Determina el cumplimiento del segundo postulado de Golomb	RESG2Z,RESG2U, resulta.txt
t-gol3.c	Determina el cumplimiento del tercer postulado de Golomb	RESG3, resulta.txt
t-clin.c	Calcula la complejidad lineal de la secuencia y el LCP	RESLCP, resulta.txt, RESLCPDESV
t-frec.c	Pasa el test estadístico de frecuencia	resulta.txt
t-par.c	Pasa el test estadístico de parejas	resulta.txt
t-tri.c	Pasa el test estadístico de tríos	resulta.txt
t-cuar.c	Pasa el test estadístico de cuartetos	resulta.txt
t-tran.c	Pasa el test estadístico de transiciones	resulta.txt
t-scorr.c	Pasa el test estadístico de correlación serie	resulta.txt
t-pok.c	Pasa el test estadístico de poker	resulta.txt
t-ru.c	Pasa el test estadístico de subsecuencias ascendentes	resulta.txt
t-rd.c	Pasa el test estadístico de subsecuencias descendentes	resulta.txt
t-perm.c	Pasa el test estadístico de permutaciones	resulta.txt
t-coup.c	Pasa el test estadístico colector de cupones	resulta.txt
t-ram.c	Pasa el test estadístico de subsecuencias por encima de la media	resulta.txt
t-rbm.c	Pasa el test estadístico de subsecuencias por debajo de la media	resulta.txt
t-lcp1.c	Pasa el test estadístico basado en el LCP (con precisión)	resulta.txt
t-lcp2.c	Pasa el test estadístico basado en el LCP (sin precisión)	resulta.txt
t-epb.c	Pasa el test estadístico de entropía por bit	resulta.txt

9.2 Uso del programa

En este apartado se describe el funcionamiento del programa y los pasos que hay que dar para obtener unos resultados válidos. Los pasos son los siguientes:

1) Lo primero que hay que hacer para testear un generador de secuencias pseudoaleatorias es simularlo mediante un programa usando algún lenguaje de alto nivel (C, FORTRAN, PASCAL, etc), de forma que genere un fichero con la secuencia de bits de salida. Este fichero debe estar constituído por ceros y unos consecutivos, pero almacenados como si fueran caracteres sucesivos y se le debe llamar *SECUEN*. Todo este proceso se puede hacer tanto en el directorio *generadores.dir*, o bien crearlo y ejecutarlo en cualquier parte. Una vez obtenido el fichero *SECUEN*, este debe ir al directorio *tests*.

2) Una consideración de gran importancia a tener en cuenta a la hora de estudiar este tipo de secuencias es si se conoce su periodo. Puesto que algunos tests descritos en el libro como los postulados de Golomb o la complejidad lineal, sólo tienen sentido si se ejecutan sobre un periodo completo de la secuencia, si no se conoce éste por métodos teóricos es necesario calcularlo antes de proseguir el estudio. Para ello, se puede utilizar el programa *t-per.c*, tal como se describe más adelante.

Si se conoce el periodo de la secuencia que se va a estudiar, el fichero *SECUEN* deberá contener, al menos, dos periodos de dicha secuencia. Esto es debido a que el perfil de complejidad lineal (LCP) de una secuencia con buenas propiedades de aleatoriedad debe crecer siguiendo la línea $k/2$, donde k es el número de bits procesados hasta ese momento, hasta alcanzar un valor lo más cercano posible al periodo. Para ello se deben procesar 2 periodos de la secuencia (evidentemente, si sabemos de antemano que la complejidad lineal final es menor que la mitad del periodo esto no será necesario, pero si no sabemos nada sobre ella a priori, será necesario disponer de al menos dos periodos de la secuencia). Si no se conoce a priori el periodo del generador (por métodos teóricos) será necesario calcularlo. Para ello existe un programa detector del periodo (*t-per.c*), cuyo programa fuente (escrito en lenguaje C) está en el directorio *tests* [PUIG92]. El fichero *batch* correspondiente, llamado *t-per.bat*, se encuentra en el directorio *batch***.** El proceso que se debe seguir es el siguiente:

a) Crear una cierta cantidad de bits de salida del generador (LONG_SEC) y actualizar este valor en el fichero *param.h* (que se encuentra en el directorio *tests*) con el número de bits de que consta nuestra secuencia. El programa detector de periodo implementado se basa en calcular la autocorrelación entre una cierta cantidad de los primeros bits de la secuencia (LONG_TEST) con el resto de la secuencia, por lo que se debe ajustar también este parámetro en el fichero *parám.h*. Por ejemplo, para una secuencia de periodo aproximado a 1.000 ó 10.000 bits (o incluso más), un buen valor para LONG_TEST puede ser 2000 300. Sin embargo, si el periodo aproximado esperado de la secuencia es por ejemplo 100, sería conveniente afinar este parámetro a 20 ó 30.

b) Ir al directorio *batch*, y ejecutar el fichero *t-per.bat.*

c) Colocar en el directorio *results* un fichero llamado *RESPER* que contendrá el periodo de la secuencia si se ha encontrado. Si se conoce el periodo, ya se podrá obtener el fichero *SECUEN*, teniendo en cuenta, tal como se ha dicho antes, que debe contener dos periodos de la secuencia. Si no se hubiera hallado el periodo, se debe volver a realizar el proceso aumentando el número de bits de la secuencia que se estudia (LONG_SEC) o aumentando el valor del parámetro LONG_TEST en el fichero *parám.h.*

A la hora de pasar el programa detector de periodo hay que tener mucho cuidado, ya que el hecho de realizar la búsqueda de esta forma puede conducir a falsas alarmas. Por ejemplo, si se dispone de una secuencia muy larga y se le da al parámetro LONG_TEST un valor pequeño, puede ser que al ir estudiando la secuencia aparezca alguna subsecuencia que coincida con ella pero, sin embargo, no constituya el comienzo de la repetición de la secuencia. Para estar seguros, lo mejor será pasar el programa detector de periodo para diferentes valores de LONG_TEST, lo que disminuirá mucho la probabilidad de falsa alarma.

9.3 Actualización inicial de parámetros y ficheros de resultados

Una vez se ha obtenido el periodo de la secuencia y se ha generado un fichero *SECUEN* que contenga, en forma de caracteres ASCCI "0" y "1", dos periodos de la secuencia de salida del generador, ya se puede proceder a la evaluación de las características del generador. El proceso que se debe seguir es el siguiente:

a) Modificar convenientemente el fichero *param.h* que se encuentra en el directorio *tests***.** Los parámetros que aparecen en este fichero son los siguientes:

-PERIODO: Periodo de la secuencia del generador bajo estudio.

-LONG_SEC: Parámetro descrito antes para el cálculo del periodo e irrelevante aquí.

-LONG_TEST: Parámetro descrito antes para el cálculo del periodo e irrelevante aquí.

-Num_datos: Número de veces que se realizará un test chi-cuadrado sobre diversas partes de la secuencia para algunos de los tests estadísticos (tests de frecuencia, parejas, tríos, cuartetos y transiciones) o, lo que es lo mismo, es el número de entradas que tendrá el test de Kolmogorov-Smirnov que decidirá sobre estos resultados. Este valor suele ser usualmente de 10.

-SUCESOS: Número de sucesos que se van a emplear para el cálculo del test de frecuencia y los tests serie (parejas, tríos, cuartetos). Por defecto, este valor está puesto a 8, aunque también se pueden emplear valores como 64, 512, etc. Al fijar este valor, hay que tener en cuenta la cantidad mímima de bits que deberá tener la secuencia para poder realizar correctamente el test según el valor que hayamos fijado. En la tabla 9.2 se muestra el mínimo número de bits de secuencia necesarios para los diversos tests en función del número de sucesos (s = 8, 64, 512).

Tabla 9.2 Bits necesarios en función del número de sucesos elegidos para los tests

TEST	*s = 8*	*s = 64*	*s = 512*
Frecuencia	16	128	1.024
Parejas	64	512	4.096
Tríos	192	1.536	12.288
Cuartetos	512	4.096	52.786
Transiciones	64	512	4.096

-MAX_L: Este parámero define el número de bloques máximo sobre el que se calculará la entropía (usando para ello el test de entropía por bit de Mauer descrito en el capítulo 5). Para decidir este parámetro hay que recurir a la tabla 9.3 en la que aparece el máximo L que se podrá

usar teniendo en cuenta la longitud de nuestra secuencia (2×PERIODO). Si se quisiera pasar este test para tamaños de bloques mayores a los que podemos, habría que generar una secuencia de periodo mayor.

-COTA: Este parámetro indica una cota inferior, a partir de la cual se van a mostrar en el fichero de resultados *resulta.txt* los picos de la función de autocorrelación (tercer postulado de Golomb) cuyo valor esté por encima de ella.

b) Una vez ajustados los parámetros descritos en el apartado anterior ya se puede realizar la evaluación del generador. Para ello hay que ir al directorio *batch* y ejecutar el proceso *predict.bat.* Notar que, según cuál sea la máquina utilizada y la longitud de la secuencia a estudiar, hay que solicitar al sistema los recursos (tanto de memoria como de tiempo) necesarios.

c) Una vez el proceso ha terminado, se deben haber generado los siguientes ficheros:

-En el directorio *results*:

- *resulta.txt*: Este es el fichero principal de resultados. Un ejemplo de él se puede ver al final de este capítulo. Los resultados que aparecen en este fichero de texto son:
  1) Resultados del primer postulado de Golomb.
  2) Resultados del segundo postulado de Golomb.
  3) Complejidad lineal.
  4) Ratios complejidad/periodo (C/P), complejidad/cota superior (C/B) y periodo/cota superior (P/B). La cota B se describirá en el apartado 9.4.
  5) Picos de la función de autocorrelación (tercer postulado de Golomb) superiores al valor COTA especificado en el programa *param.h* (por defecto está a 0,7).
  6) Resultados de los tests estadísticos, mostrando para cada test los resultados de los *Num_datos* tests chi-cuadrado realizados sobre diversas partes de la secuencia y el resultados del test de Kolmogorov-Smirnov realizado sobre ellas, dando una indicación de si pasa el test o no. Los tests realizados cuyos resultados se pueden encontrar en este fichero se pueden ver en la tabla 9.1.

-RESG2Z: Este fichero muestra los resultados de aplicar la primera condición del segundo postulado de Golomb sobre las subsecuencias de ceros. Para ello, se muestra para cada longitud de subsecuencias la desviación entre los valores obtenidos y los que teóricamente predice este postulado.

- RESG2U: Este fichero es similar al anterior pero para las subsecuencias de unos.

- RESG3: Este fichero muestra los resultados de aplicar el tercer postulado de Golomb, es decir, es la autocorrelación normalizada de la secuencia.

-RESLCP: Este fichero muestra el perfil de complejidad lineal (LCP) de la secuencia.

-RESLCPDESV: Este fichero muestra la desviación de los valores obtenidos en el LCP sobre los valores medios calculados que se esperan.

Los ficheros de resultados RESG2Z, RESG2U, RESG3, RESLCP y RESLCPDESV representan gráficas que muestran los puntos en forma X-Y. Para visualizarlos se puede utilizar cualquier programa de visualización de gráficas.

A) Cálculo de la correlación cruzada de dos secuencias

Existe otro programa que calcula la correlación cruzada normalizada entre dos secuencias (llamadas *SECUEN1* y *SECUEN2* respectivamente). Este programa se llama *t-ccn.c* y se encuentra en el directorio *tests*, mientras que su fichero *batch* correspondiente se llama *t-ccn.bat* y se encuentra en el directorio *batch***.** Este programa funciona independientemente de los anteriores, y se puede usar siempre que se necesite calcular la correlación cruzada de dos secuencias (por ejemplo, dos secuencias de salida de un generador, obtenidas a partir de distintos estados iniciales de las celdas). Su funcionamiento es el siguiente:

a) Crear dos secuencias que se llamen *SECUEN1* y *SECUEN2* respectivamente (ambas deben ser de igual longitud) y ponerlas en el directorio *tests***.**

b) Actualizar el valor PERIODO en el fichero *parám.h* con la longitud de estas secuencias.

c) Ir al directorio *batch* y ejecutar el proceso *t-ccn.bat.*

d) El programa dará el resultado en el fichero *RESCCN* y lo colocará en el directorio *results*. Este fichero de resultados está en formato de puntos X-Y, con lo que se puede ver con un programa de visualización de gráficos.

Si lo que se desea es calcular la función de correlación cruzada no normalizada entre ambas secuencias, se puede utilizar el programa *t-ccnn.c* (en el directorio *tests*), cuyo fichero batch se denomina *t-ccnn.bat* y está en el directorio *batch*. El resultado de este programa es el mismo que en el caso normalizado pero los valores que se obtienen en la correlación no están divididos por el periodo de la secuencia y se llama *RESCCNN*. Además, el proceso que se debe seguir es el mismo que en el caso anterior. Una vez obtenidos todos estos resultados, corresponde al usuario el evaluarlos para dar una apreciación final de las propiedades del generador testeado. En el apartado siguiente se muestran algunas consideraciones que se deben tener en cuenta a la hora de evaluar estos resultados.

9.4 ¿Qué información revelan estos programas?

Si bien las bases teóricas de todos los tests que contiene el programa PREDICTEST se describen en el capítulo 5, en este apartado se describen algunas conclusiones que se pueden obtener sobre las propiedades del generador a la vista de los resultados obtenidos en los ficheros anteriores.

Primer postulado de Golomb: Este postulado dice que en un periodo de una secuencia pseudoaleatoria, la diferencia entre el número de unos y ceros no debe exceder de uno. Esto no se cumple casi nunca exactamente en la realidad (excepto para los registros de desplazamiento de realimentación lineal (LFSR)) pero, sin embargo, se puede considerar un buen comportamiento respecto a este postulado si esta diferencia no es excesivamante grande.

Segundo postulado de Golomb: Este postulado establece que en cada periodo de una secuencia pseudo aleatoria, la mitad de las subsecuencias de unos debe tener longitud 1, una cuarta parte longitud 2, una octava parte longitud 3, etc. En general, deben haber $1/2^i$ subsecuencias de longitud *i*. Lo mismo debe ocurrir con las subsecuencias de ceros. En los ficheros *RESG2U* y *RESG2Z* se muestran las desviaciones que se obtienen respecto a este postulado. Lo ideal, evidentemente, sería que esta desviación fuese nula, pero en la práctica casi siembre aparecen ciertas desviaciones sobre todo para subsecuencias de longitudes cortas. Esta desviación casi siempre disminuirá rápidamente a medida que las subsecuencias se hacen mayores. Además, el segundo postulado también establece que debe haber el mismo número total de subsecuencias de ceros que de unos.

Tercer postulado de Golomb: Este postulado establece que la función de autocorrelación normalizada de una secuencia pseudoaleatoria debe ser bivalor, teniendo un pico en el origen de valor uno y un valor muy pequeño e igual en todos los restantes puntos. Esto no se suele dar exactamente en la realidad excepto para el LFSR, pero en la práctica debemos comprobar que el resultado obtenido para la secuencia estudiada se acerca lo máximo posible a esto, mostrando un pico de valor uno en el origen (esto se cumplirá siempre), y que no presenta picos de elevado valor para el resto de los desplazamientos.

El resultado de aplicar este test se muestra en el fichero *RESG3*. Además, el programa puede mostrar los picos que estén por encima de un cierto valor que le especifiquemos (COTA en el fichero *resulta.txt*) y que por defecto es 0,7. El parámetro COTA se puede modificar, ya que se encuentra en el fichero *param.h* en el directorio *tests*.

La complejidad lineal: La complejidad lineal de una secuencia es la longitud del registro de desplazamiento con realimentación lineal (LFSR) más corto que es capaz de generar la secuencia dada, es decir, determina qué porción de subsecuencia es necesaria para ser capaz de reproducir toda la secuencia. Por tanto, para que una secuencia sea impredictible, debe presentar un valor final de la complejidad lineal del orden del periodo. Esto se puede ver además en el ratio complejidad/periodo (C/P) del fichero *resulta.txt*, que debe estar lo más cercano que sea posible al 100%.

El perfil de complejidad lineal: Para poder usar la complejidad lineal como parámetro de medida de la impredictibilidad de la secuencia, hay que estudiar la forma de crecimiento que muestra la complejidad lineal al ir procesando los bits de información de la misma.

Si se representa la complejidad lineal en función del número de bits calculados (*k*) hasta ese momento (fichero *RESLCP*), se debe ver un crecimiento siguiendo la línea de *k*/2 (aunque de forma irregular, con saltos de media 2 en altura y 4 en horizontal) hasta alcanzar el valor final de la complejidad que, como se ha visto antes, debe estar lo más ceraca posible al periodo. Para calcular esta gráfica se requieren al menos dos periodos de la secuencia. Además, el fichero *RESLCPDESV* muestra

la desviación de los valores que se van obteniendo de complejidad a medida que se procesa la secuencia, con respecto al valor esperado calculado teóricamente y que se vio en el capítulo 6. Evidentemente esta desviación debería ser idealmente nula, pero en la práctica deberemos comprobar sea pequeña.

Aleatoriedad y tests estadísticos: Los tests estadísticos se usan para detectar posibles defectos estadísticos en los generadores de bits pseudoaleatorios, por ejemplo, para detectar cuándo el modelo estadístico que describe el comportamiento del generador se desvía significativamente del de una fuente simétrica binaria. Los tests que se han pasado sobre la secuencia están descritos en el capítulo 5, y los resultados de los programas dan una indicación de si pasa o no el test. En algunos tests (frecuencia, parejas, tríos, cuartetos y transiciones) se muestra el resultado de aplicar *Num_data* tests chi-cuadrado y el resultado de aplicar un test de Kolmogorov-Smirnov sobre sus resultados, de forma que el resultado de este último test indica si la secuencia bajo estudio pasa o no el test estadístico en cuestión.

Tabla 9.3 Bits de secuencia necesarios para estudiar la entropía por bit

Longitud de bloque (L)	*Bits de secuencia necesarios*
1	10.020
2	20.080
3	30.240
4	40.640
5	51.600
6	63.840
7	78.960
8	100.480
9	136.080
10	202.400
11	335.280
12	611.520
13	1.194.960
14	2.433.760
15	5.065.200
16	10.645.760
17	22.452.240
18	47.365.920
19	99.804.720
20	209.915.200

En cualquier caso, estaremos tanto más seguros de la bondad de un generador cuanto mayor sea el número de tests estadísticos de este tipo que pase.

Además, el hecho de que un generador pase un test no indica que no vaya a fracasar totalmente al intentar pasar otro.

Test de entropía por bit: Este test ofrece principalmente dos ventajas sobre los tests estadísticos anteriores. En primer lugar, a diferencia de ellos, este test es capaz de detectar cualquiera de los diversos defectos estadísticos que pueda presentar la secuencia de salida del generador, incluídos todos los que detectaban los tests mencionados anteriomente. Además, es capaz de medir determinados parámetros (como por ejemplo la frecuencia relativa de unos o ceros) y mide la significatividad del defecto.

Para realizar esto, este test calcula la entropía por bit de la secuencia. Para llevar a cabo el test se procede a partir de las secuencias en bloques disjuntos (no superponibles) de longitud *L*. Este test se pasará para bloques de longitud 1 hasta MAX_L, donde éste es un parámetro que hay que especificar en el fichero *param.h* antes de pasar el test y estará limitado en función de la cantidad de bits que contenga el periodo que hemos generado.

Sin embargo, en general, la entropía obtenida irá creciendo a medida que crezca el valor de *L* y, para un buen generador, debería acercarse lo máximo posible al valor ideal 1. El programa da los resultados de aplicar este test para los tamaños de *L* especificados (1...MAX_L).

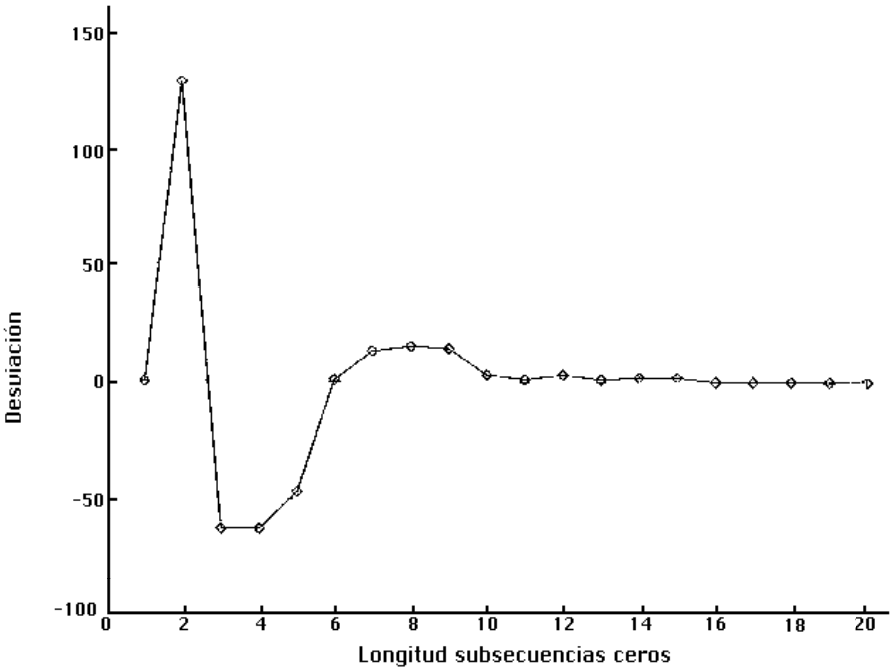
La tabla 9.3 muestra hasta qué longitud de bloque puede estudiarse en función de la longitud (PERIODO) de la secuencia bajo estudio, y teniendo en cuenta que al parámetro *K* se le da un valor de 10.000.

9.5 Ejemplo de utilización

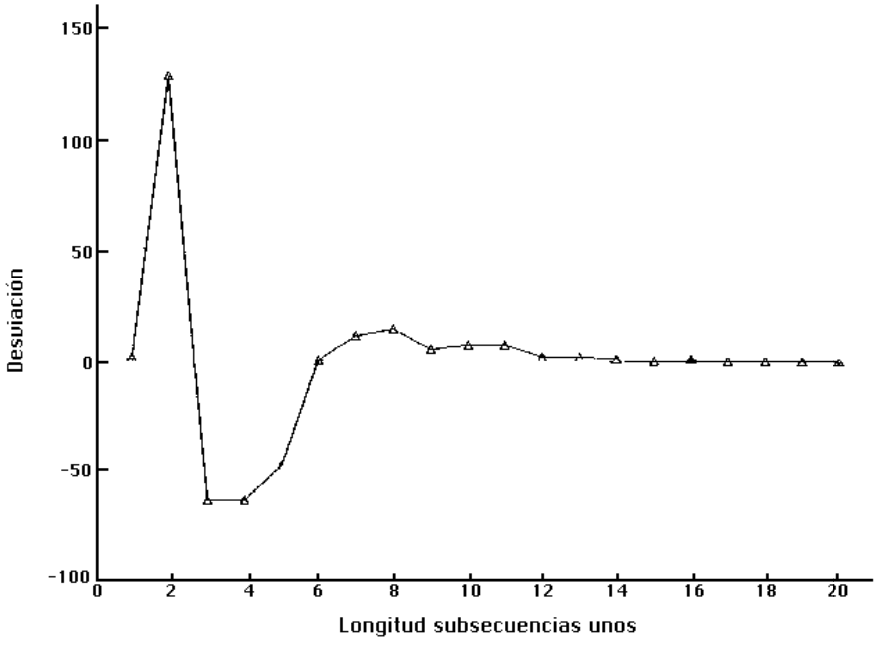
Para mostrar de forma gráfica el funcionamiento del programa se ha analizado un generador de secuencias pseudoaleatorias propuesto por S.W. Jennings en 1980 [JEN80] basado en combinar las secuencias de salida de dos LFSRs mediante un multiplexor, que ya se estudió en el capítulo 4. Veamos cómo este programa nos permite obtener unas conclusiones sobre la aleatoriedad de las secuencias producidas por el generador. Para ello, se ha empleado el modelo que se muestra en la figura 9.1 para generar la secuencia bajo estudio (*SECUEN*). El programa que implementa este generador se llama J*ennings.c*, y se puede encontrar en el directorio *generado*. Un análisis más detallado de este generador se puede encontrar en [CRU92][CRU93P] y [CRU93G].

Usando el detector de periodo, se obtiene que el periodo de la secuencia es de 14.329 bits, lo que coincide con el estudio teórico realizado por Jennings que demuestra que si los polinomios de los LFSR son primitivos y tienen *m* y *n* celdas respectivamente, el periodo de la secuencia de salida es $P = (2^m -1)(2^n -1) = (2^3-1)(2^{11} -1) = 14.329$. Analicemos ahora los resultados obtenidos de aplicar PREDICTEST sobre la secuencia de salida de este generador. Para evaluar mejor las dimensiones de algunos resultados, es conveniente definir una cota superior $B = 2^L = 2^{14} = 16.384$, que es la máxima longitud de la secuencia que podría obtenerse con una longitud de clave de $m+n = 14$ bits. Si comparamos esta cota con el periodo obtenido, sale un rendimiento P/B del 87,45 % que, si bien es un valor bastante bueno, aunque todavía lejano del valor máximo que podría obtenerse.

A continuación se muestran los ficheros de resultados obtenidos y un breve análisis de ellos. Veamos en primer lugar los resultados obtenidos en cuanto al cumplimiento de los postulados de Golomb.



Gráfica 9.1 Representación gráfica del fichero RESG2Z



Gráfica 9.2 Representación gráfica del fichero RESG2U

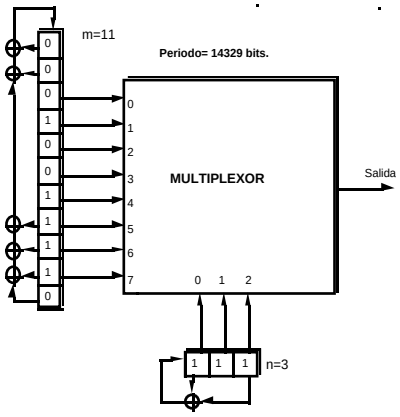


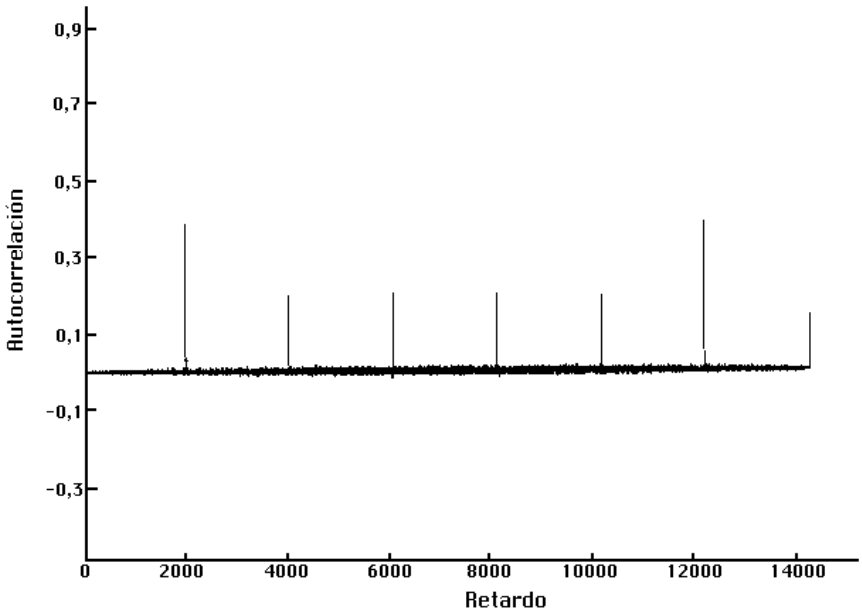
Figura 9.1 Generador de Jennings

El primer postulado nos dice que en un periodo de la secuencia, la diferencia entre el número total de ceros y el número total de unos no debe exceder de uno. Tal como muestra el fichero *resulta.txt* que se muestra al final de esta capítulo, en un periodo de salida de este generador se tienen 7.168 unos y 7.161 ceros con lo que la diferencia es de 7.

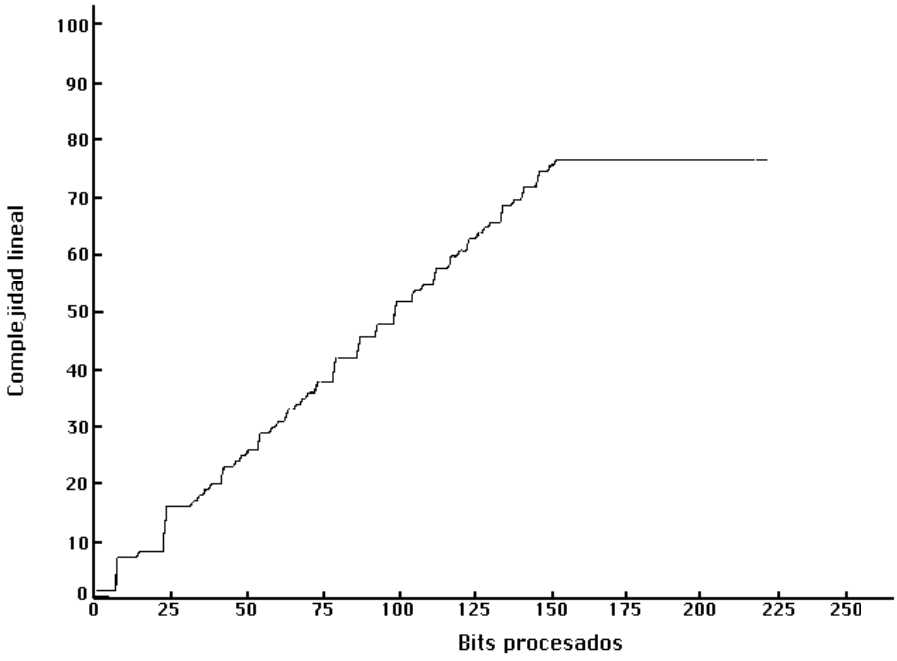
Si bien no cumple estrictamente este postulado, la diferencia es bastante pequeña. Con estos resultados sale una probabilidad para los unos de 0,5002, y una probabilidad para los ceros de 0,4998, por lo que están bastante equilibrados.

El segundo postulado establece dos condiciones. En primer lugar, el número total de subsecuencias de unos debe ser igual al número total de subsecuencias de ceros. Para este generador salen 3.583 subsecuencias de ceros y 3.584 subsecuencias de unos. La diferencia que sale, de una sola subsecuencia, se puede considerar despreciable. La segunda condición es que tanto para ceros como para unos, deben haber $(1/2)^i$ subsecuencias para cada longitud *i*. La mejor forma de verlo es poniendo en una gráfica la desviación de los resultados obtenidos respecto a la teoría. Esto se puede ver en el fichero *RESG2Z* (representado en la gráfica 9.1) y RESG2U (representado en la gráfica 9.2). En estas gráficas se puede observar que tanto para unos como para ceros la diferencia respecto a lo que establece el segundo postulado es apreciable para subsecuencias de longitudes menores que 10, y es muy pronunciada para longitudes de subsecuencias entre 2 y 5. Para longitudes de subsecuencias mayores que 10, los resultados coinciden con los valores esperados por el segundo postulado. Hay que notar que valores negativos de desviación significan que hay una carencia de subsecuencias para una determinada longitud, mientras los valores positivos reflejan un exceso para esa longitud. Además, en el fichero *resulta.txt* se puede ver la diferencia entre el número total de subsecuencias de ceros y unos para cada longitud.

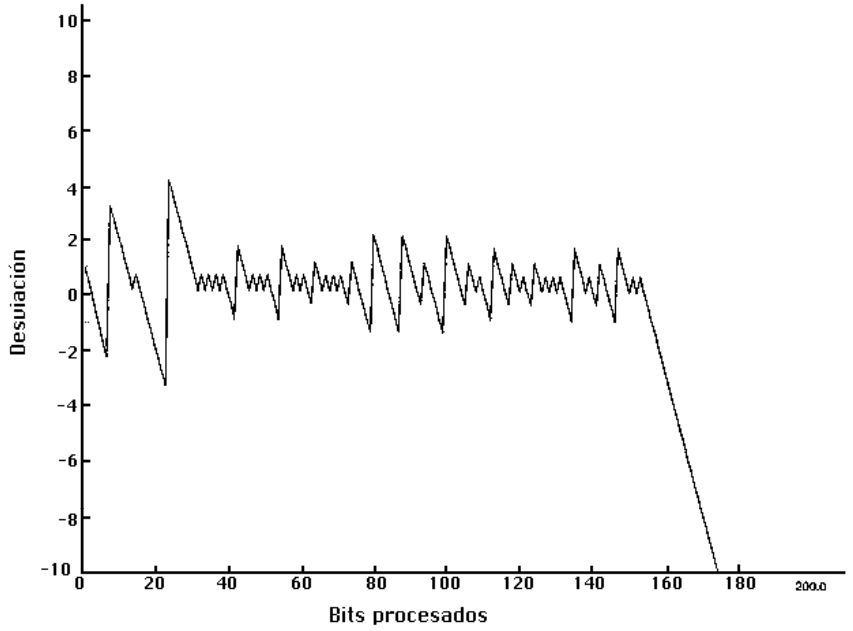
El tercer postulado establece que la función de autocorrelación, calculada a lo largo de un periodo de la secuencia debe ser bivalor. El fichero *RESG3*, que se muestra en la gráfica 9.3, muestra el resultado obtenido. Se observan unos picos de autocorrelación fuera de fase que, si se analizan cuidadosamente, resultan ser sumamente peligrosos si se utiliza el generador en aplicaciones como el cifrado en flujo, ya que dan información de la estructura interna del generador. Veámoslo:



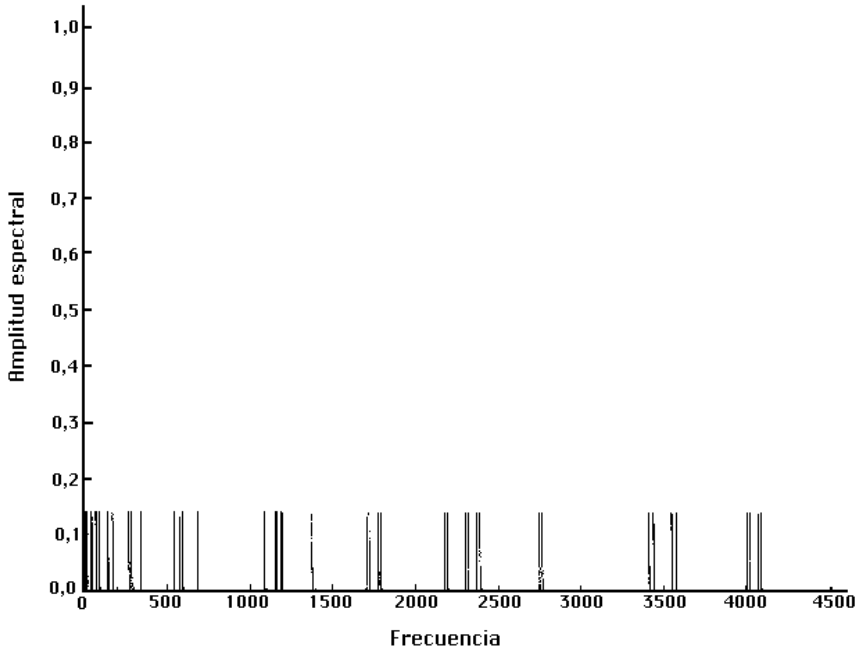
Gráfica 9.3 Representación gráfica del fichero RESG3



Grafica 9.4 Representación gráfica del fichero RESLCP



Grafica 9.5 Representación gráfica del fichero RESLCPDESV



Grafica 9.6 Representación gráfica del fichero ESPECTRAL

Los picos de la gráfica 9.3 aparecen centrados en los siguientes valores: 0, 2.047, 4.094, 6.141, 8.188, 10.235, 12.282, 14.329. Si se observan estos números, el primero y el último son los que corresponden al desplazamiento cero (valores en fase). Pero entre ellos hay 7 intervalos separados por otros picos de autocorrelación fuera de fase bastante altos. Esto corresponde con 2^3-1, que es el periodo de uno de los dos LFSR que usa el generador. Además, los picos están separados 2.047 entre sí, y este valor corresponde a 2^{11} -1 que corresponde con el periodo del otro LFSR que usa el generador.

El fichero *RESLCP*, representado en la gráfica 9.4, muestra el perfil de complejidad lineal obtenido usando el algoritmo de Massey-Berlekamp que está implementado en el fichero *t-clin.c* del directorio *tests*. Tal como se espera, este perfil sigue la línea de $k/2$ (aunque irregularmente) para secuencias de longitud k, por lo que en este sentido tiene un buen comportamiento. El problema es que se estabiliza después de que el algoritmo haya procesado tan sólo 144 bits de la secuencia, resultando un valor final de complejidad lineal $C = 77$, que se mantiene hasta el final del periodo, por lo que es un valor peligrosamente bajo si se quiere emplear este generador en aplicaciones de cifrado en flujo. Este mal comportamiento se puede apreciar mejor en las relaciones C/P y C/B que salen del 0,53 % y del 0,47 % respectivamente. En dicha gráfica se ve que la desviación respecto al LCP medio es pequeña hasta el valor 77, pero lógicamente, al estabilizarse la complejidad en este valor, la desviación comienza a crecer apreciablemente.

En el fichero *RESLCPDESV*, que se ha representado en la gráfica 9.5, se muestra la desviación entre el perfil de complejidad que se ha obtenido de la secuencia bajo estudio y el perfil de complejidad lineal ideal esperado para una secuencia aleatoria de dicha longitud.

Este generador se comporta bastante bien en cuanto a la estadística, ya que la secuencia producida por el generador ha pasado prácticamente todos los tests estadísticos a los que se ha sometido su secuencia de salida. En el fichero *resulta.txt* se pueden ver los resultados de los tests estadísticos a los que ha sido sometido la secuencia, la indicación de si los pasa o no, y algunos resultados significativos para cada uno de los tests. En este fichero, también se puede observar el resultado del test de entropía que, con la longitud dada, sólo se ha podido pasar para tamaños de bloque de longitud máxima 4. Se puede observar que un resultado en torno a 0,65 en la entropía por bit no es un valor excesivamente bueno, por lo que no podemos afirmar de forma rotunda que las secuencias producidas por el generador muestren un grado aceptable de aleatoriedad.

El fichero *ESPECTRAL*, que se ha representado en la gráfica 9.6, muestra el comportamiento del generador respecto al test espectral. En él se observan algunos picos (aunque no demasiado elevados) que no deberían aparecer. El lector debe notar que PREDICTEST no tiene en la versión actual ningún programa que implemente el test espectral, por lo que un ejercicio interesante puede consistir en realizar este programa.

Como conclusión del análisis de los ficheros de resultados obtenidos del programa PREDICTEST se puede mencionar su buen comportamiento en cuanto a periodo, al primer postulado de Golomb, a la primera condición del segundo postulado y a los tests estadísticos. Sin embargo, las secuencias producidas no muestran un buen comportamiento con respecto a la segunda condición del segundo postulado de Golomb. Los resultados de los tests muestran además otros problemas: el primero es que la complejidad lineal obtenida es muy baja frente a la longitud del periodo. Esto hace que la secuencia sea bastante predecible. El segundo problema es que en la función de autocorrelación aparecen unos picos que, si se analizan detenidamente, dan una información muy peligrosa si el

generador se emplea en una aplicación como el cifrado en flujo. Finalmente, en el test espectral aparecen picos indeseables que apartan el comportamiento de la secuencia bajo estudio del de una secuencia verdaderamente aleatoria. Estos dos problemas hacen poco aconsejable el uso del generador de Jennings por sí solo en ciertas aplicaciones como la criptografía. A continuación se muestra un listado del fichero *resulta.txt* que muestra los resultados comentados anteriormente.

9.6 Listado del fichero *resulta.txt*

```
/*****************************************************************************/
/***************                                                ***************/
/***************              Fichero resulta.txt               ***************/
/***************                                                ***************/
/*****************************************************************************/

----------PRIMER POSTULADO DE GOLOMB----------------------------------------------------

    Número de unos en 14.329 bits procesados =  7.168
    Numero de ceros en 14.329 bits procesados = 7.161
    NO CUMPLE el primer postulado de Golomb

----------SEGUNDO POSTULADO DE GOLOMB---------------------------------------------------

    Número total de subsecuencias de ceros =     3.583
    Número total de subsecuencias de unos  =     3.584
    -------------------------
    Longitudes subsecuencias de 1 cero =         1.791
    Longitudes subsecuencias de 1 uno =          1.793
    Longitudes subsecuencias de 2 ceros =        1.024
    Longitudes subsecuencias de 2 unos =         1.024
    Longitudes subsecuencias de 3 ceros =          384
    Longitudes subsecuencias de 3 unos =           384
    Longitudes subsecuencias de 4 ceros =          160
    Longitudes subsecuencias de 4 unos =           160
    Longitudes subsecuencias de 5 ceros =           64
    Longitudes subsecuencias de 5 unos =            64
    Longitudes subsecuencias de 6 ceros =           56
    Longitudes subsecuencias de 6 unos =            56
    Longitudes subsecuencias de 7 ceros =           40
    Longitudes subsecuencias de 7 unos =            39
    Longitudes subsecuencias de 8 ceros =           28
    Longitudes subsecuencias de 8 unos =            28
    Longitudes subsecuencias de 9 ceros =           20
```

Longitudes subsecuencias de 9 unos = 12
Longitudes subsecuencias de 10 ceros = 6
Longitudes subsecuencias de 10 unos = 10
Longitudes subsecuencias de 11 ceros = 2
Longitudes subsecuencias de 11 unos = 8
Longitudes subsecuencias de 12 ceros = 3
Longitudes subsecuencias de 12 unos = 2
Longitudes subsecuencias de 13 ceros = 1
Longitudes subsecuencias de 13 unos = 2
Longitudes subsecuencias de 14 ceros = 2
Longitudes subsecuencias de 14 unos = 1
Longitudes subsecuencias de 15 ceros = 2
Longitudes subsecuencias de 15 unos = 0
Longitudes subsecuencias de 16 ceros = 0
Longitudes subsecuencias de 16 unos = 1
Longitudes subsecuencias de 17 ceros = 0
Longitudes subsecuencias de 17 unos = 0
Longitudes subsecuencias de 18 ceros = 0
Longitudes subsecuencias de 18 unos = 0
Longitudes subsecuencias de 19 ceros = 0
Longitudes subsecuencias de 19 unos = 0

----------TERCER POSTULADO DE GOLOMB----------

(Picos de la función de autocorrelación con valor > 0,7)
0 1,000000

----------COMPLEJIDAD LINEAL----------

COMPLEJIDAD LINEAL = 77 (el algoritmo da solución única)
Polinomio característico: 1000000100000000000000000000100000000000000010
00000100000000000000000000001000000
Cota P/B = 87,45% Cota C/P = 0,53% Cota C/B = 0,46%

----------Resultados del TEST DE FRECUENCIA para 8 sucesos----------

Test Chi^2_0 función chi = 2,000000 correcto = 0
Test Chi^2_1 función chi = 0,500000 correcto = 0
Test Chi^2_2 función chi = 2,000000 correcto = 0
Test Chi^2_3 función chi = 0,500000 correcto = 0
Test Chi^2_4 función chi = 0,500000 correcto = 0
Test Chi^2_5 función chi = 0,000000 correcto = 1
Test Chi^2_6 función chi = 2,000000 correcto = 0

Test Chi^2_7 función chi = 2,000000 correcto = 0
Test Chi^2_8 función chi = 0,500000 correcto = 0
Test Chi^2_9 función chi = 2,000000 correcto = 0
Test de KS (5%-95%) = *PASA* el test de frecuencia (45%)

----------Resultados del TEST DE PAREJAS para 8 sucesos----------

Test Chi^2_0 función chi = 1,000000 correcto = 0
Test Chi^2_1 función chi = 3,000000 correcto = 0
Test Chi^2_2 función chi = 5,000000 correcto = 0
Test Chi^2_3 función chi = 5,000000 correcto = 0
Test Chi^2_4 función chi = 5,000000 correcto = 0
Test Chi^2_5 función chi = 2,000000 correcto = 0
Test Chi^2_6 función chi = 1,000000 correcto = 0
Test Chi^2_7 función chi = 3,000000 correcto = 0
Test Chi^2_8 función chi = 3,000000 correcto = 0
Test Chi^2_9 función chi = 3,000000 correcto = 0
Test de KS (5%-95%) = *PASA* el test de parejas (50%)

----------Resultados del TEST DE TRIOS para 8 sucesos----------

Test Chi^2_0 función chi = 4,000000 correcto = 0
Test Chi^2_1 función chi = 8,000000 correcto = 0
Test Chi^2_2 función chi = 6,000000 correcto = 0
Test Chi^2_3 función chi = 14,000000 correcto = 0
Test Chi^2_4 función chi = 22,000000 correcto = 1
Test Chi^2_5 función chi = 8,000000 correcto = 0
Test Chi^2_6 función chi = 2,000000 correcto = 1
Test Chi^2_7 función chi = 12,000000 correcto = 0
Test Chi^2_8 función chi = 10,000000 correcto = 0
Test Chi^2_9 función chi = 8,000000 correcto = 0
Test de KS (5%-95%) = PASA el test de tríos (48%)

----------Resultados del TEST DE CUARTETOS para 8 sucesos----------

Test Chi^2_0 función chi = 16,000000 correcto = 0
Test Chi^2_1 función chi = 12,000000 correcto = 0
Test Chi^2_2 función chi = 20,000000 correcto = 0
Test Chi^2_3 función chi = 20,000000 correcto = 0
Test Chi^2_4 función chi = 24,000000 correcto = 0
Test Chi^2_5 función chi = 24,000000 correcto = 0
Test Chi^2_6 función chi = 8,000000 correcto = 0
Test Chi^2_7 función chi = 12,000000 correcto = 0

Test Chi^2_8 función chi = 28,000000 correcto = 1
Test Chi^2_9 función chi = 8,000000 correcto = 0
Test de KS (5%-95%) = *PASA* el test de cuartetos (36%)

----------Resultados del TEST DE TRANSICIONES para 8 sucesos----------

Test Chi^2_0 función chi = 2,000000 correcto = 0
Test Chi^2_1 función chi = 4,000000 correcto = 0
Test Chi^2_2 función chi = 1,000000 correcto = 0
Test Chi^2_3 función chi = 6,000000 correcto = 1
Test Chi^2_4 función chi = 6,000000 correcto = 1
Test Chi^2_5 función chi = 3,000000 correcto = 0
Test Chi^2_6 función chi = 2,000000 correcto = 0
Test Chi^2_7 función chi = 2,000000 correcto = 0
Test Chi^2_8 función chi = 2,000000 correcto = 0
Test Chi^2_9 función chi = 4,000000 correcto = 0
Test de KS (5%-95%) = *PASA* el test de transiciones (47%)

----------TEST DE CORRELACION SERIE ----------

Coeficiente de correlación serie (C) = -0,000768
Cota inferior = -0,016777 Cota superior = 0,016637
Para pasar el test C debe caer entre las cotas superior e inferior
Por tanto, la secuencia *PASA* el test de correlación serie

---------- TEST DE POKER ----------

Hay 0 repokers
Hay 41 entre fulls y pokers
Hay 282 entre dobles parejas y tríos
Hay 496 parejas
Hay 137 grupos con las 5 cartas diferentes
Sobre estos resultados se ha pasado un test chi-cuadrado con 4 grados de libertad
cuyo resultados es: *NO PASA* el test de Poker (Queda por debajo del 1%)

----------TEST DE SUBSECUENCIAS ASCENDENTES----------

Número de subsecuencias crecientes de longitud 1 = 603
Número de subsecuencias crecientes de longitud 2 = 864
Número de subsecuencias crecientes de longitud 3 = 385
Número de subsecuencias crecientes de longitud 4 = 94
Número de subsecuencias crecientes de longitud 5 = 13
Número de subsecuencias crecientes de longitud 6 o más = 1

Sobre estos resultados se aplica un test chi-cuadrado con 6 grados de libertad con el resultado: *NO PASA* el test de subsecuencias ascendentes (Queda por debajo del 1%)

----------TEST DE SUBSECUENCIAS DESCENDENTES----------

Número de subsecuencias decrecientes de longitud 1 = 616
Número de subsecuencias decrecientes de longitud 2 = 874
Número de subsecuencias decrecientes de longitud 3 = 384
Número de subsecuencias decrecientes de longitud 4 = 85
Número de subsecuencias decrecientes de longitud 5 = 12
Número de subsecuencias decrecientes de longitud 6 o más = 3
Sobre estos resultados se aplica un test chi-cuadrado con 6 grados de libertad con el resultado: *NO PASA* el test de subsecuencias descendentes (Queda por debajo del 1%)

----------TEST DE PERMUTACIONES----------

Se divide la secuencia en 1.318 grupos de 3 elementos y se cuenta el número de posibles ordenaciones. Sobre estos resultados se pasa un test chi-cuadrado con 5 grados de libertad cuyo resultado es: *PASA* el test de permutaciones

----------TEST COLECTOR DE CUPONES----------

Número de observaciones = 20
Hay 0 segmentos de longitud 8 y se esperaban 0,048065
Hay 0 segmentos de longitud 9 y se esperaban 0,168228
Hay 0 segmentos de longitud 10 y se esperaban 0,346971
Hay 0 segmentos de longitud 11 y se esperaban 0,551999
Hay 1 segmento de longitud 12 y se esperaban 0,750866
Hay 1 segmento de longitud 13 y se esperaban 0,920285
Hay 0 segmentos de longitud 14 y se esperaban 1,047947
Hay 1 segmento de longitud 15 y se esperaban 1,130591
Hay 2 segmentos de longitud 16 y se esperaban 1,171006
Hay 0 segmentos de longitud 17 y se esperaban 1,175287
Hay 1 segmento de longitud 18 y se esperaban 1,150803
Hay 3 segmentos de longitud 19 y se esperaban 1,104906
Hay 0 segmentos de longitud 20 y se esperaban 1,044212
Hay 0 segmentos de longitud 21 y se esperaban 0,974282
Hay 1 segmento de longitud 22 y se esperaban 0,899557
Hay 3 segmentos de longitud 23 y se esperaban 0,823431
Hay 1 segmento de longitud 24 y se esperaban 0,748388
Hay 0 segmentos de longitud 25 y se esperaban 0,676163
Hay 0 segmentos de longitud 26 y se esperaban 0,607892
Hay 0 segmentos de longitud 27 y se esperaban 0,544255

Hay 0 segmentos de longitud 28 y se esperaban 0,485586
Hay 0 segmentos de longitud 29 y se esperaban 0,431974
Hay 6 segmentos de longitud 30 y se esperaban 3,197309

Sobre estos resultados se aplica un test chi-cuadrado con 23 grados de libertad con el resultado: *PASA* el test colector de cupones

----------TEST DE SUBSECUENCIAS POR ENCIMA DE LA MEDIA----------

Se han computado 50 gaps (número de observaciones).
Número de brechas de longitud 0 = 31
Número de brechas de longitud 1 = 8
Número de brechas de longitud 2 = 4
Número de brechas de longitud 3 = 4
Número de brechas de longitud 4 = 2
Número de brechas de longitud 5 = 0
Número de brechas de longitud 6 o mayor = 1

Sobre estos resultados se aplica un test chi-cuadrado con 7 grados de libertad con el resultado: *PASA* el test de subsecuencias por encima de la media

----------TEST DE SUBSECUENCIAS POR DEBAJO DE LA MEDIA----------

Se han computado 50 gaps (número de observaciones).
Número de brechas de longitud 0 = 28
Número de brechas de longitud 1 = 7
Número de brechas de longitud 2 = 7
Número de brechas de longitud 3 = 2
Número de brechas de longitud 4 = 2
Número de brechas de longitud 5 = 2
Número de brechas de longitud 6 o mayor = 2

Sobre estos resultados se aplica un test chi-cuadrado con 7 grados de libertad con el resultado: *PASA* test de subsecuencias por debajo de la media

---------TEST DE NUMERO DE SALTOS EN EL LCP----------

Número de saltos esperados en el LCP de una secuencia de 14.329 bits = 7.164,916667
Número de saltos encontrados en el LCP = 39 *NO PASA* el test del número de saltos en el LCP

---------TEST DE ALTURA DE LOS SALTOS DEL LCP----------

NO PASA el test estadístico de altura de saltos en el LCP (Queda por debajo del 1%)

----------RESULTADOS DEL TEST DE ENTROPIA----------

L = 1,	Q = 20,	K = 10000,	ENTROPIA = 0,623700
L = 2,	Q = 40,	K = 10000,	ENTROPIA = 0,653250
L = 3,	Q = 80,	K = 10000,	ENTROPIA = 0,637800
L = 4,	Q = 160,	K = 10000,	ENTROPIA = 0,501175

9.7 Bibliografía

[CRU92] CRUSELLES, E.; MELÚS, J. L; PUIG, M. *Evaluación del Generador de Jennings y su Aplicación en el Sistema de Acceso Condicional de la Norma MAC/packet.* II Reunión Española sobre Criptología. Madrid, Octubre 1992.

[CRU93P] CRUSELLES, E. *Evaluación de la Seguridad del Sistems de Acceso Condicional Eurocrypt para la Protección de Transmisiones de Imagen, Voz y Datos Vía Satélite*. Proyecto Fin de Carrera. Universidad Politécnica de Cataluña. 1993.

[CRU93G] CRUSELLES, E.; MELÚS, J. L. *An Overview of Security in Eurocrypt Conditional Access System*. GLOBECOM´93. Houston, Texas. Noviembre 1993.

[CRU93B] CRUSELLES, E.; MELÚS, J. *Estudio Comparativo de Cifradores de Flujo.* Informe Final Beca CESCA. Centro de Supercomputación de Cataluña y Universidad Politécnica de Cataluña, Barcelona, Diciembre de 1993.

[CRU94C] CRUSELLES, E.; MELÚS, J. *Use of Supercomputing in Cryptography*. Curso High Performance Computing: Current Trends and Applications. Centro de Supercomputación de Cataluña, Barcelona, 21-22 de Noviembre de 1994.

[JEN80] JENNINGS, S. M. *A Special Class of Binary Sequences*. Ph. D. Thesis. Westfield College, London University, 1980.

[PUI92] PUIG, M. *Estudio Comparativo de Cifradores en Flujo. Aplicaciones en Comunicaciones Móviles*. Proyecto fin de Carrera. Universidad Politécnica de Cataluña. 1992.

[CES93] *Estudio Comparativo de cifradores de Flujo*. Recull de Projectes de Recerca del CESCA. Consell Científic del CESCA. B-14.576-1993. pp. 271-272. Abril 1992.

Indice alfabético

T

U

V

W

Z