Paralelización en la interpolación de la decodificación por listas de códigos Reed-Solomon

M.ª de los Ángeles Simarro-Haro [1] Jose Moreira [2] Marcel Fernández Muñoz [3] Miquel Soriano [4] A. González [5] F. J. Martínez-Zaldívar [6]

Resumen— **Los códigos Reed-Solomon son códigos correctores de errores ampliamente utilizados en numerosos sistemas de comunicación y almacenamiento digital. Guruswami y Sudan propusieron una estrategia de decodificación por lista de estos códigos, ampliada y mejorada posteriormente por Koetter y Vardy para el caso de un modelo de canal con decisiones *soft* o *indecisas* a su salida. Este algoritmo se basa en dos procesos principales: el proceso de la interpolación y el de factorización. En el presente artículo se muestran análisis de rendimiento de la paralelización del primero de los procesos, que implica la mayor carga computacional del algoritmo total.**

Palabras clave—**Reed-Solomon, decodificación de lista, Koetter-Vardy, Guruswami-Sudan, computación paralela, multinúcleo, watermarking, fingerprinting**

I. Introducción

LOS códigos Reed-Solomon (RS) son una clase de códigos correctores de errores utilizados desde su concepción hasta la actualidad en numerosas aplicaciones. Podríamos citar algunos ejemplos como la difusión de televisión digital o Digital Video Broadcasting (DVB) [9], WIMAX [8], etc. Otras aplicaciones como *fingerprinting* o *watermarking* se benefician de esta familia de códigos y de sus propiedades, manteniendo consistentemente la información oculta que puede identificar a usuarios deshonestos que distribuyan de forma no autorizada contenidos digitales [15], [6].

La decodificación de los códigos RS ha sido un tema activo de investigación en las decadas pasadas. El último avance significativo llegó con el trabajo de Sudan [2], donde se presentó el primer algoritmo de decodificación de lista para códigos RS. La decodificación de lista fue introducida por Elias en [5], tal y como su nombre sugiere, proporciona a la salida del decodificador no una única palabra código sino un conjunto de ellas, entre las cuales se encuentra la palabra código que más verosímilmente fue transmitida, ampliando de esta manera la capacidad correctora de errores de dicho código. Este trabajo inicial más tarde fue extendido por Guruswami y Sudan en [1]. Por otro lado, en [3], Koetter y Vardy usan la información de fiabilidad (*soft*) proporcionada por el canal para mejorar el proceso de decodificación.

A. Objetivos y estructura del artículo

En la bibliografía podemos encontrar implementaciones paralelas del algoritmo de decodificación de Koetter-Vardy (algoritmo KV) pero principalmente para implementaciones *hardware* en FPGA (véase [11], [12], [13]). Desafortunadamente, y según el conocimiento de los autores, no existen hasta la fecha en la bibliografía implementaciones y variaciones de este algoritmo sobre todo en el contexto de sistemas paralelos multinúcleos. Estos sistemas tienen un amplio uso en dispositivos móviles, por lo que una eficiente implementación paralela basada en un paradigma de programación multihilo puede resultar de interés. En este artículo se presenta la paralelización del proceso de interpolación, el cual supone la parte más compleja computacionalmente del algoritmo KV. Después de describir su implementación secuencial, procedemos a presentar el esquema de paralelización y la evaluación del rendimiento para finalmente extraer conclusiones del trabajo realizado.

II. Algoritmo de decodificación de Koetter-Vardy

Antes de exponer el algoritmo KV, vamos a definir formalmente los códigos RS: sea $GF(q)$ un cuerpo finito de orden q, y γ un elemento primitivo del mismo. El código Reed-Solomon de longitud $n = q - 1$ y dimensión k sobre $GF(q)$, denotado como RS[n, k], se define como el siguiente subespacio vectorial de $GF(q)^n$:

$$\mathrm{RS}[n,k] = \{(p(\gamma^1), \ldots, p(\gamma^n)) : p(x) \in GF(q)[x]_{k-1}\},$$

donde $GF(q)[x]_{k-1}$ es el anillo de polinomios sobre $GF(q)$ de grado menor que k.

La palabra código es transmitida a través de un canal ruidoso y el decodificador recibe una versión corrupta de ella. En este trabajo, nos centramos en la decodificación de lista de los códigos RS haciendo uso del algoritmo KV [3].

A. Algoritmo KV

En esta sección hacemos una breve revisión del algoritmo KV. Para una descripción más detallada

[1]Instituto de Telecomunicaciones y Aplicaciones Multimedia (iTEAM), Universitat Politècnica de València, Spain, e-mail: mdesiha@teleco.upv.es

[2]Departament de Enginyeria Telemàtica, Universitat Politècnica de Catalunya, Spain, e-mail: jose.moreira@entel.upc.edu

[3]Departament de Enginyeria Telemàtica, Universitat Politècnica de Catalunya, Spain, e-mail: marcel@entel.upc.edu

[4]Departament de Enginyeria Telemàtica, Universitat Politècnica de Catalunya, Spain, e-mail: soriano@entel.upc.edu

[5]Instituto de Telecomunicaciones y Aplicaciones Multimedia (iTEAM). Universitat Politècnica de València, Spain, e-mail: agonzal@dcom.upv.es

[6]Instituto de Telecomunicaciones y Aplicaciones Multimedia (iTEAM), Universitat Politècnica de València, Spain, e-mail: fjmartin@dcom.upv.es

véase [3].

El algoritmo KV está basado en el algoritmo de Guruswami-Sudan. Ambos algoritmos consisten en dos pasos principales: un paso de interpolación y otro de factorización. Adicionalmente, KV incluye un proceso previo de preprocesado en el cual la información de fiabilidad proporcionada por el canal se traduce en el conjunto de restricciones de interpolación usadas para el siguiente proceso. La información de fiabilidad dada al decodificador usualmente toma la forma de probabilidad de que un símbolo dado haya sido enviado.

Antes de presentar el algoritmo KV, cabe señalar que si $Q(x,y) = \sum_{i,j} q_{i,j} x^i y^j$ es un polinomio bivariable con coeficientes en $GF(q)$, el $(1, k-1)$-grado ponderado de $Q(x,y)$ es definido como $\text{máx}\{i + (k-1)j : q_{i,j} \neq 0\}$. El proceso del algoritmo de KV para un código RS[n, k] es como sigue:

1. *Paso de preprocesado*: traduce la información *soft* proporcionada por el canal en un conjunto de restricciones de interpolación: $S = \{(x_l, y_l, m_l)\}$, donde $x_l, y_l \in GF(q)$ y m_l es un entero positivo.
2. *Paso de Interpolación*: construye un polinomio bivariable sobre $GF(q)$ de mínimo $(1, k-1)$-grado ponderado,

$$Q(x,y) = \sum_{i=0}^{d_x} \sum_{j=0}^{d_y} a_{i,j} x^i y^j, \qquad (1)$$

   tal que satisface las restricciones de interpolación. Esto es, $Q(x,y)$, tiene un cero en (x_l, y_l) de multiplicidad m_l para cada tupla $(x_l, y_l, m_l) \in S$. Esto es equivalente a especificar el valor de la función y sus $m_l - 1$ derivadas del polinomio de interpolación.
3. *Paso de Factorización*: encuentra todos los factores de $Q(x,y)$ de la forma $y - p(x)$, donde $p(x)$ es un polinomio de grado $k-1$ o menos. La salida del algoritmo es la lista de palabras código $\mathcal{L}$ generada a partir de cada $p(x)$.

Podemos encontrar varios algoritmos que resuelven el proceso de interpolación (véase por ejemplo [4], [14]). De todos ellos, se ha implementado el algoritmo de Koetter el cual puede ser encontrado en [7], [4]. Existen también diferentes algoritmos para el proceso de factorización necesario para completar el algoritmo de decodificación [4], [10], [16], [17]. En este caso, el algoritmo Roth-Ruckenstein ha sido el implementado y puede ser encontrado en [4].

B. El paso de interpolación

Dado que nuestro objetivo en el presente trabajo es la aceleración del paso de interpolación cuando se ejecuta en entornos multinúcleo, presentamos con mayor detalle esta parte del algoritmo de KV.

Algoritmo de Interpolación de Koetter

Entrada: Conjunto de restricciones
$S = \{(x_i, y_i, m_i)\}$; máximo y-grado dy;
orden del monomio $(1, v)$.
Salida: Polinomio bivariable $Q(x,y)$ de mínimo $(1, k-1)$-grado ponderado que satisface las restricciones de interpolación.
Initialización: $Q_j = y^j$ para $j = 0$ hasta dy
Proceso:

```
Para cada (x_i, y_i, m_i) ∈ S
    para (r, s) = (0, 0) hasta (m_i − 1, 1)
        para j = 0 hasta dy
            Δ_j = coeff(Q_j(x + x_i, y + y_i), x^r y^s)
        j* = argmin_(1,k−1){Q_j : j ∈ J}
        f = Q_j
        Δ = Δ_j*
            if (j ≠ j*)
                Q_j = ΔQ_j − Δ_j f
            else (j = j*)
                Q_j = (x − x_i) f
Q(x, y) = min_(1,k−1){Q_j(x, y)}
```

Para un polinomio bivariable $Q(x,y) = \sum_{i,j} q_{i,j} x^i y^j$, la función $\text{coeff}(Q(x,y), x^r \ y^s)$ simplemente devuelve $q_{r,s}$, lo cual es empleado para calcular la denominada *discrepancia*. También las funciones $\min_{(1,k-1)}$ y $\text{argmin}_{(1,k-1)}$ devuelven el mínimo $(1, k-1)$-grado ponderado y el índice del polinomio de mínimo $(1, k-1)$-grado ponderado, respectivamente.

III. Implementación Secuencial

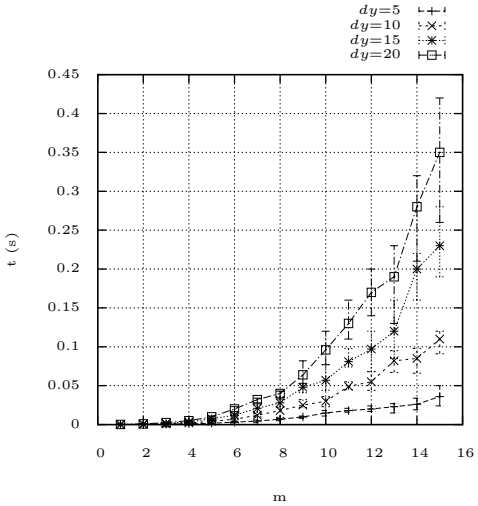
A. Plataforma hardware/software

Todos los test han sido ejecutados en un procesador SMP de Intel(R) con dos Xeon (TM) de seis núcleos X5675 @3.07 GHz con 128 GB de memoria principal, con el *hyperthreading* desactivado, ejecutados en un sistema operativo Linux con un *kernel* 2.6.32, utilizando el compilador de C/C++ de Intel `icc`, versión 12.1.

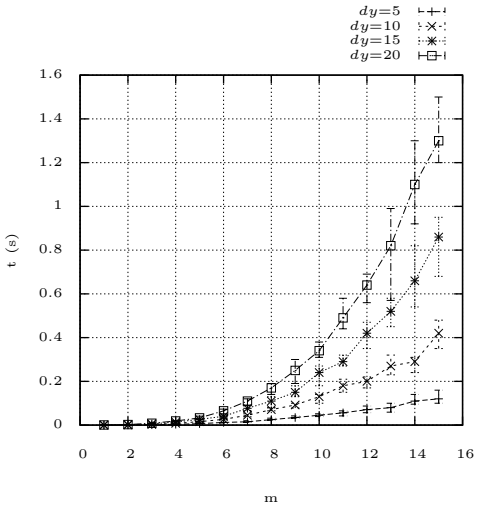
B. Tiempo de ejecución

Las figuras 1(a) y 1(b) muestran el tiempo de ejecución del proceso de interpolación respecto a la multiplicidad m de los puntos para varios valores máximos del grado de y (dy), para diferente número de puntos respectivamente. Las mediciones han sido promediadas con 10 muestras temporales, y los valores máximo y mínimo de estas mediciones se indican en el segmento vertical dibujado sobre las mismas. En las gráficas podemos observar cómo el tiempo de ejecución crece con el valor de la multiplicidad m, con el máximo grado de y, dy y con el número de puntos n_p.

La proporción de tiempo de ejecución invertida en el proceso de interpolación con respecto al tiempo de decodificación total (tiempo de interpolación más factorización) proporciona una información muy importante para justificar la estrategia de la paralelización; esto es mostrado en las figuras 2(a) y 2(b). Un

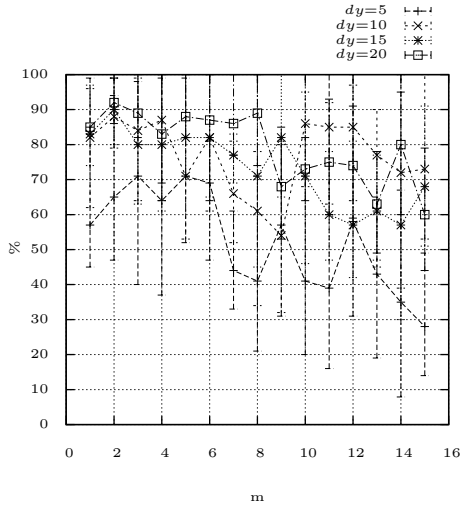


(a) $n_p = 10$

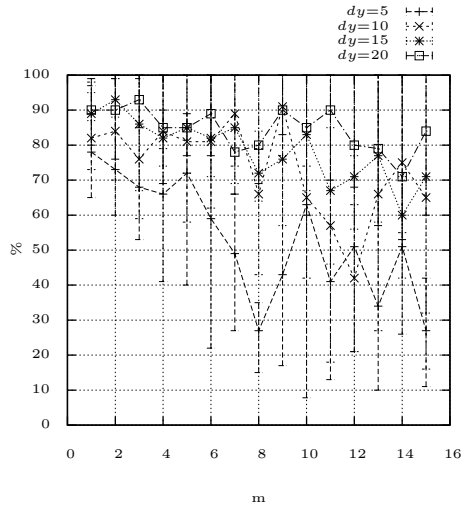


(b) $n_p = 20$

Fig. 1. Tiempo de ejecución secuencial del algoritmo de interpolación



(a) $n_p = 10$



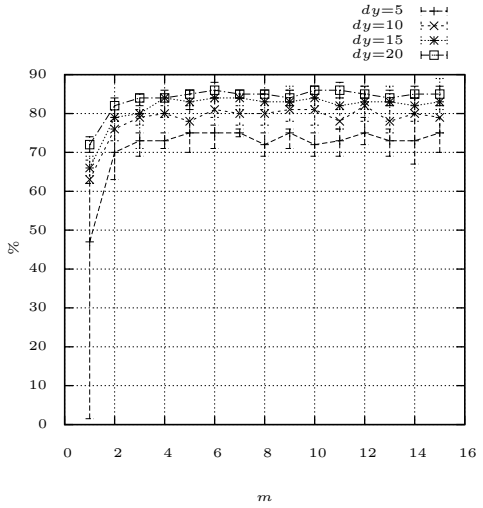
(b) $n_p = 20$

Fig. 2. Proporción del tiempo de ejecución del algoritmo secuencial de interpolación respecto al tiempo total de ejecución del algoritmo de decodificación

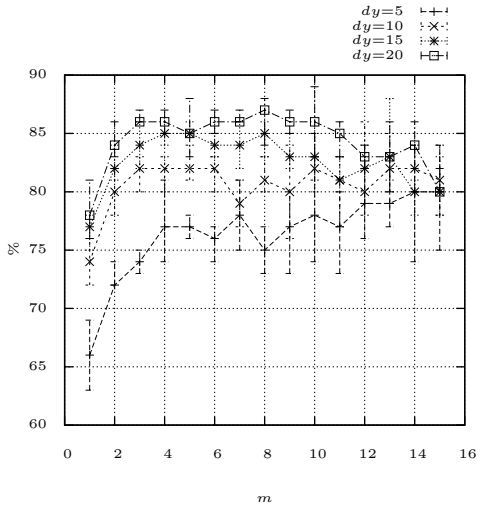
gran porcentaje de tiempo de ejecución del algoritmo de decodificación KV es concentrado en la parte de interpolación tal y como muestran las curvas. En general, valores mayores para *dy* proporcionan mayor proporción del tiempo de ejecución a la interpolación. La varianza en la proporción del tiempo de medida es alta debido a una fuerte dependencia con la instancia concreta del problema. Por ello desde un punto de vista de la eficiencia del futuro algoritmo paralelo, es interesante concentrar el esfuerzo de la paralelización en esta parte del algoritmo.

Dentro del proceso de interpolación, cada iteración del algoritmo de Koetter tiene dependencia de datos con la iteración anterior, por lo que una paralelización de cualquiera de los bucles `for` externos tendría un rendimiento notablemente ineficiente. En el bucle más interno controlado por la variable j, en el cual las *discrepancias* son calculadas, cada iteración es completamente independiente de cualquier otra, pudiéndose paralelizar de manera eficiente. Afortunadamente, el resto del código es relativamente rápido con respecto a este bucle interior. La figuras 3(a) y 3(b) muestran la ratio entre el tiempo de ejecución invertido en el cálculo de la discrepacia con respecto al tiempo total de ejecución del proceso de interpolación. Esta proporción es relativamente constante (entre el 75 y el 85 %) con respecto al valor de la mutliplicicdad m y el número de puntos n_p, y más elevado conforme aumenta el valor de dy.

De nuevo, esta parte del código es la que más tiempo consume, por ello su paralelización puede potenciar una aceleración favorable en el cálculo. De cualquier modo, esta fracción serie del algoritmo limitarán el incremento de velocidad o *speedup* máximo alcanzable en el rango de 4–6,6 debido la ley de Amdahl.

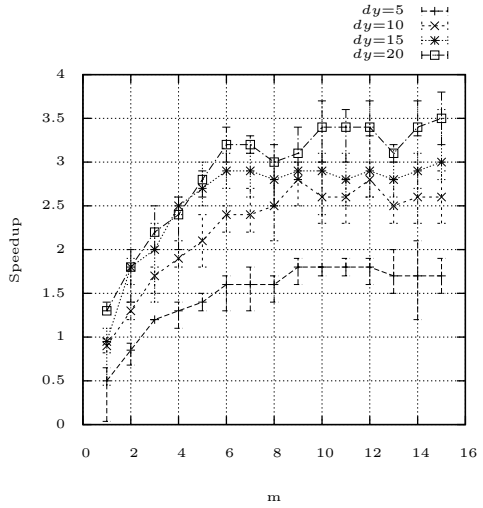


(a) $n_p = 10$

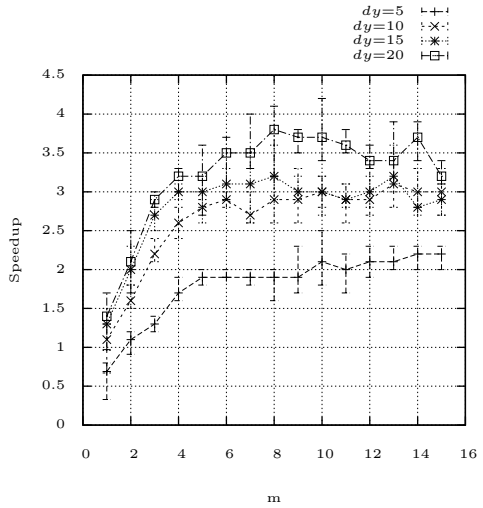


(b) $n_p = 20$

Fig. 3. Proporción del tiempo de ejecución en el cálculo de la discrepancia respecto al tiempo de ejecución de la interpolación



(a) $n_p = 10$



(b) $n_p = 20$

Fig. 4. Speedup en el algoritmo de interpolación de Koetter

IV. Paralelización y speedup

Hemos usado directivas de compilación *OpenMP* para paralelizar el bucle interno del algoritmo Koetter. Se han obtenido tiempos secuenciales y paralelos para el proceso de interpolación; en todos los casos se ha empleado la opción de optimización `-O3` del compilador.

Las figuras 4(a) y 4(b) muestran el *speedup* eel código de interpolación para dos cantidades de puntos de interpolación distintas. El rango de speedup obtenido varía aproximadamente entre 1,7 y 3,5 y depende de los valores de los parámetros de la ejecución. Si se comparan estos valores con el máximo posible proporcionado por la ley de Amdahl la eficiencia se encuentra aproximadamente en el intervalo del 43-53 %.

V. Conclusiones

En este artículo se han mostrado los resultados del rendimiento obtenido tras la paralelización del proceso de interpolación del algoritmo de Koetter-Vardy. En este algoritmo encontramos bucles anidados con dependencia de datos entre las iteraciones, por lo que a priori la paralelización previsible es ineficiente. La única parte del algoritmo que puede ser completamente paralelizada es el bucle más interno; en cualquier caso, debido a una significativa fracción serie del algoritmo, el máximo límite en el speedup alcanzable viene impuesto por la ley de Amdahl. Esta es la razón principal por la cual sólo se ha conseguido un moderado speedup o moderada eficiencia en la paralelización.

Agradecimientos

El presente proyecto ha sido financiado por el Vicerrectorado de Investigación de la UPV a través del programa de Apoyo a la Investigación y Desarrollo (PAID-05-11-2733), por la Generalitat Valenciana a

través del proyecto PROMETEO/2009/012, por el Gobierno español a través de proyectos Consolider Ingenio CSD2007-00004 'ARES" Y TEC2011-26491 "COPPI", y el gobierno catalán bajo la concesión 2009 SGR-1362. J. Moreira es el receptor de una FPU, AP2009-3854, del Ministerio de Educación de España.

Referencias

[1] V. Guruswami and M. Sudan, *Improved decoding of Reed-Solomon codes and algebraic geometric codes*, IEE Transactions on Information Theory, vol. 45, no. 6 pp. 1757-1767, 1999.

[2] M. Sudan, *Decoding of Reed-Solomon codes beyond the error correction bound*, J. Complexity, vol 12, pp. 180-193, 1997.

[3] R. Koetter and A. Vardy, *Algebraic soft-decision decoding of Reed-Solomon codes*, IEE Transactions on Information Theory, vol. 49, no. 11, pp. 2809-2825, 1999.

[4] Tood K. Moon, *Alternate Decoding Algorithms for Reed Solomon Codes*, Wiley, 2005.

[5] Peter Elias, *List Decoding for noisy channels*, Technical report, Laboratory of electronics Massachusets Institute of Technology, September 1957.

[6] M.Fernández and M. Soriano, *Fingerprinting concatenated codes with efficient identification*, Lecture Notes in Computer Science, 2433: 459-470, 2002.

[7] R.J McEllice, *The Guruswami-Sudan Decoding Algorithm for Reed-Solomon Codes*, Technical report, California Institute of Technology, Pasadena, California, May 2003.

[8] S. Lujan, S. Almagro, A. Cabrera, J. Suardiaz and F. Cerdan, *Códigos RS y su aplicación a la capa física 802.16 en FPGAs*, Universidad Politécnica de Cataluña.

[9] M. Francis and R. Green, *Forward Error Correction in Digital Television Broadcast System*, Xilinx v1.0, September 5, 2007.

[10] Daniel Augot and Lancelot Pequet, *An alternative to Factorization: a Speed for Sudan's decoding algorithm and its generalization to algebraic-geometric codes*, Intitut National de Recherche en Informatique et en Automatique, October 1998 INRIA no. 3532.

[11] Laurier Bouulianne and Warrer J. Gross *SIMD Implementation of Interpolation in Algebraic Soft-Decision Reed-Solomon Decoding*, Signal Processing Systems Design and Implementation pp.750-755, November 2005.

[12] Warren J. Gross, Frank R. Kschischang and P. Glenn Gulak, *Architecture and Implementation of an Interpolation Processor for Soft-Decision Reed-Solomon Decoding*, IEEE Transactions on very large scale integration (VLSI) Systems, Vol. 15, No. 3, March 2007.

[13] Bainan Chen and Xinmiao Zhang, *FPGA implementation of a factorization processor for soft-decision Reed-Solomon decoding*, Circuits and Systems, pp. 944 - 947, May 2008.

[14] Kwankyu Lee and Michael E. O'Sullivan, *An Interpolation Algorithm using Göbner Bases for Soft-Decision Decoding of Reed-Solomon Codes*, Information Theory, pp. 2032 - 2036, July 2006.

[15] A. Barg, G. R. Blakley and G. A. Kabatiansky, *Digital fingerprinting codes: Problem statements, constructions, identification of traitors*, Trans. Inf. Theory, pp. 852–865, (49) 2003.

[16] X. W. Wu and P. H. Siegel, *Efficient root-finding algorithm with application to list decoding of algebraic geometric codes*, IEEE Trans. Inform. Theory, Vol. 47, No. 9, (September 2001), pp. 2579-2587.

[17] R. Nielsen and T. Høholdt, *Decoding Reed-Solomon codes beyond half the minimum distance*, Coding Theory, Cryptography and Related Areas, Eds. Berlin, Germany: Springer-Verlag, 2000, pp. 221-236.