# Diseño de Códigos Correctores de Errores con Algoritmos Genéticos

E. Alba, J. F. Chicano, B. Dorronsoro, G. Luque

*Resumen*— **Algunos sistemas de telecomunicaciones no pueden tolerar el coste de la repetición de un mensaje cuando se corrompe por el camino. En su lugar, el mensaje debe ser "corregido" de alguna forma en el destino. En estos casos es adecuado el uso de un *código corrector de errores*. El problema de encontrar un código corrector de errores de $n$ bits y $M$ palabras que corrija un máximo número de errores dado es NP-Completo. Por esta razón, el problema se ha resuelto en la literatura con técnicas heurísticas tales como Recocido Simulado y Algoritmos Genéticos. En este artículo abordamos el problema con algoritmos genéticos en panmixia, estructurados e hibridados con un nuevo algoritmo de búsqueda local hecho a medida para el problema. Los resultados demuestran que las técnicas distribuidas e híbridas obtienen los mejores resultados.**

*Palabras clave*— **Teoría de la información, Heurísticos, Búsqueda local, Hibridación**

## I. Introducción

ALGUNOS sistemas de telecomunicaciones no pueden tolerar el reenvío de un mensaje que se ha corrompido en el camino. Ejemplos de estos sistemas son la telefonía, la televisión y la radio digitales. En todas estas aplicaciones, la información debe ser entregada con bajo retraso, por lo que usar un código corrector de errores es adecuado para evitar sobrecargas por retransmisión. Existen algunas comunicaciones punto a punto donde la repetición de un mensaje corrupto puede no ser un inconveniente, aunque en las comunicaciones en tiempo real suele serlo, especialmente en comunicaciones multipunto (radio, televisión, etc.).

Para estos sistemas hay muchos tipos de códigos; nosotros nos centramos aquí en los códigos correctores de errores (ECC) binarios lineales de bloques [1]. Estos códigos están formados por un conjunto de palabras, cada una de las cuales es una cadena binaria de cierta longitud. Los mensajes están compuestos por una secuencia de palabras. Cuando se transmite un mensaje, las cadenas binarias que lo forman son enviadas a través del medio físico formando un flujo de bits que puede sufrir modificaciones conforme atraviesa el medio. Cuando un bit de una palabra ha cambiado en el camino, decimos que hay un error en esa palabra. Al otro lado llegará una palabra incorrecta. Si queremos garantizar que se puedan detectar hasta $e$ errores, la distancia de Hamming entre cualquier par de palabras debe ser al menos $e+1$. Por otro lado, si la distancia de Hamming entre cada par de palabras es mayor o igual a $2e+1$, las cadenas con $e$ o menos errores pueden corregirse. La corrección se consigue asignando a la cadena incorrecta su palabra de código más cercana en el destino.

Así que, cuanto más separadas estén las palabras del código, más errores pueden ser corregidos. Dada la longitud de las palabras y el número de palabras de un código, el problema de encontrar un código óptimo es NP-Completo [2].

Las técnicas exactas pueden ser útiles para pequeñas instancias, pero para instancias más realistas son inviables por su alto tiempo de cómputo. Por esta razón, debemos usar técnicas heurísticas tales como Recocido Simulado (SA), Algoritmos Genéticos (GA), etc. ([2], [3]). En este artículo abordamos el problema con varias técnicas heurísticas (detalles en la Sección III).

El artículo está organizado como sigue. En la Sección II se presenta el problema de Diseño de Códigos Correctores de Errores. Después, introducimos todos los algoritmos empleados para resolver el problema (Sección III). En la Sección IV describimos las pruebas realizadas y los resultados obtenidos. Finalmente, en la Sección V presentamos las conclusiones y el trabajo futuro.

## II. El Problema de Diseño de Códigos Correctores de Errores (ECC)

Un problema básico en la construcción de cualquier sistema de comunicación es encontrar un código adecuado para transmitir mensajes a través de un canal con ruido de la forma más rápida y fiable posible. Como suele ocurrir en optimización, debemos optimizar varios objetivos en conflicto para conseguir esto. El primer objetivo es encontrar palabras con longitud mínima, para que el mensaje se pueda transmitir rápidamente. Por otro lado, la distancia de Hamming entre palabras debe ser máxima para garantizar un alto nivel de corrección en el receptor, lo que sugiere

E.T.S.I. Informática. Campus de Teatinos. UMA.
e-mail: eat@lcc.uma.es
e-mail: chicano@lcc.uma.es
e-mail: dorronsoro@uma.es
e-mail: gabriel@lcc.uma.es

incluir alta redundancia (palabras más largas). Cuanto menor sea la longitud de las palabras, menor es la mínima distancia de Hamming entre ellas. La idea es la siguiente: si todas las palabras están separadas por $d$ bits, cualquier modificación de hasta $\lfloor (d-1)/2 \rfloor$ bits en una palabra válida puede ser corregida tomando la palabra más cercana (esquema en la Fig. 1).

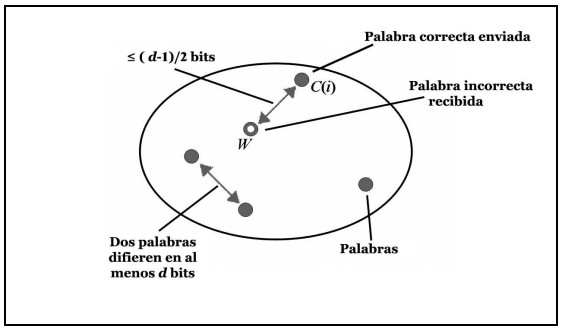


Fig. 1. Interpretación gráfica de un sistema corrector.

Los códigos binarios lineales de bloques pueden ser representados como un vector de tres parámetros $(n, M, d)$, donde $n$ es el número de bits de cada palabra del código, $M$ es el número de palabras y $d$ es la mínima distancia de Hamming entre cualquier par de palabras $C(i)$ y $C(j)$ $(i \neq j)$.

Un código óptimo es aquél que maximiza $d$, dados $n$ y $M$. En este artículo, resolvemos el problema para un código de 24 palabras y 12 bits $(M = 24, n = 12)$ como en [2]. A modo ilustrativo, el espacio de búsqueda es $\binom{2^n}{M}$, que en nuestro caso significa aproximadamente $10^{63}$ códigos diferentes. La búsqueda exhaustiva está claramente descartada. Esta instancia del problema se ha usado con frecuencia en la literatura por su dificultad. En [2] se usan de 256 a 16384 elementos de procesamiento de una máquina SIMD para resolverlo. En [4] la instancia es abordada con GAs distribuidos y en panmixia (una única población centralizada) concluyendo la superioridad del enfoque distribuido descentralizado. Para esta instancia, un código óptimo es aquél con $d = 6$ [5].

## III. Algoritmos

Para resolver el problema hemos empleado algoritmos genéticos, un nuevo algoritmo de búsqueda local (Algoritmo de Repulsión) y varios algoritmos híbridos entre ellos. A continuación los describiremos brevemente.

### A. Algoritmos Genéticos

Los Algoritmos Genéticos (GAs) [6] son métodos estocásticos de búsqueda que han sido aplicados con éxito en muchos problemas de búsqueda, optimización y aprendizaje máquina. A diferencia de muchas otras técnicas de optimización, los GAs mantienen una población de soluciones tentativas codificadas (individuos) que son manipuladas competitivamente mediante operadores de variación para encontrar un óptimo global.

Un GA procede de un modo iterativo generando nuevos individuos a partir de los antiguos. En la Fig. 2 presentamos el pseudocódigo de un GA general. Comienza creando una población aleatoria $P(0)$ de $\mu$ individuos, cada uno codifica las $p$ variables del problema, normalmente como un vector sobre $\mathbb{B} = \{0, 1\}$ $(I = \mathbb{B}^{p \cdot l_x})$ o $\mathbb{R}$ $(I = \mathbb{R}^p)$. Se necesita una función de evaluación $\Phi$ para asociar a cada individuo un valor real indicando su conveniencia para el problema (fitness). El algoritmo canónico aplica operadores estocásticos tales como selección, recombinación y mutación sobre la población para calcular una generación completa de nuevos individuos (genGA) o un individuo en cada paso (ssGA). Cada operador tiene un conjunto de parámetros que determinan su comportamiento. Estos parámetros se representan como subíndices de los operadores en la Fig. 2. Cuando se cumple el criterio de parada $\iota$ el algoritmo se detiene. La solución es el mejor individuo encontrado.

$t := 0$;
inicializar: $P(0) := \{\vec{a}_1(0), \ldots, \vec{a}_\mu(0)\} \in I^\mu$;
evaluar: $P(0) : \{\Phi(\vec{a}_1(0)), \ldots, \Phi(\vec{a}_\mu(0))\}$;
**while** $\iota(P(t)) \neq$ **true do**
  seleccionar: $P'(t) := s_{\Theta_s}(P(t))$;
  recombinar: $P''(t) := \otimes_{\Theta_c}(P'(t))$;
  mutar: $P'''(t) := m_{\Theta_m}(P''(t))$;
  evaluar: $P'''(t) : \{\Phi(\vec{a}_1'''(t)), \ldots, \Phi(\vec{a}_\lambda'''(t))\}$;
  reemplazar: $P(t+1) := r_{\Theta_r}(P'''(t) \cup Q)$;
  <**fase de comunicación**>
  $t := t + 1$;
**end while**

Fig. 2. Pseudocódigo de un GA general.

El algoritmo CHC [7] es un algoritmo genético no tradicional que utiliza una estrategia de selección muy conservadora: elige siempre a los mejores $\mu$ individuos para formar parte de la nueva población. Aplica además un operador de recombinación muy explorador $(HUX)$ que produce una descendencia lo más diferente posible a ambos padres. CHC fue desarrollado con la idea de solucionar los problemas debidos a la convergencia prematura que sufren los algoritmos genéticos y, de hecho, incorpora un mecanismo para reiniciar el algoritmo ante una condición de convergencia prematura. También introduce un sesgo para evitar el cruce de individuos similares.

Los GAs descentralizados son propensos al paralelismo, ya que las operaciones sobre los individuos pueden fácilmente realizarse en paralelo. La estructuración de la población contribuye al

mantenimiento de la diversidad, lo que favorece la convergencia a soluciones óptimas. Existen evidencias de esta gran diversidad y eficiencia [8] así como de sus capacidades multi-solución.

Dentro de los GAs descentralizados podemos distinguir los distribuidos (dGAs) y los celulares (cGAs). Los primeros se caracterizan por el pequeño número de subpoblaciones de gran tamaño y su bajo grado de acoplamiento (migraciones esporádicas). Los segundos se caracterizan por la existencia de un gran número de subpoblaciones con pocos individuos (normalmente uno) y su alto grado de acoplamiento.

### *B. Algoritmo de Repulsión*

Para mejorar los resultados existentes, hemos desarrollado un nuevo algoritmo de búsqueda local para el problema ECC: el Algoritmo de Repulsión (RA). Este algoritmo está basado en la Física, concretamente en la Electrostática.

La idea central es que, si colocamos un conjunto de partículas cargadas con la misma carga (y signo) en la superficie de una esfera, tenderán a alejarse y separarse unas de otras. Las fuerzas que producen esta configuración pueden calcularse usando la Ley de Coulomb. Las palabras de un código binario se pueden ver como partículas en un espacio $n$-dimensional, donde $n$ es la longitud de las palabras. Una solución dada para el ECC es mejor que otra cuando las distancias de Hamming entre las palabras es mayor en la primera solución. Así que, la idea es considerar las palabras como partículas y aplicar la Ley de Coulomb para calcular las fuerzas entre ellas; entonces, podríamos simular su movimiento libre (repulsión combinada) para alcanzar un estado de baja energía.

Existen algunas diferencias entre ambos escenarios. Para empezar, las palabras del código se encuentran en los vértices de un hipercubo de $n$ dimensiones (o $n$-cubo) mientras que las partículas están confinadas en la superficie de una esfera de tres dimensiones. Las dimensiones no son un problema puesto que podemos generalizar la Ley de Coulomb a espacios de $n$ dimensiones. Pero las partículas se pueden mover con total libertad sobre la superficie de una hiperesfera (o $n$-esfera) mientras que las palabras sólo pueden moverse por las aristas de un $n$-cubo para acabar en otro vértice adyacente.

RA calcula las fuerzas entre cada par de palabras del código (considerando que tienen la misma carga) para después calcular la fuerza resultante que se aplica a cada una de ellas. Llamemos $\mathbf{f}_{ij}$ a la fuerza que la palabra $j$ ejerce sobre la $i$ y $\mathbf{F}_i$ a la fuerza resultante que se aplica sobre la palabra $i$-ésima (Fig. 3). Las expresiones de estos vectores son:

$$\mathbf{f}_{ij} = \frac{1}{d_{ij}} \frac{\mathbf{p}_i - \mathbf{p}_j}{\sqrt{d_{ij}}} \tag{1}$$

$$\mathbf{F}_i = \sum_{j=1, j\neq i}^{M} \mathbf{f}_{ij} \tag{2}$$

donde $\mathbf{p}_i$ y $\mathbf{p}_j$ son las palabras $i$ y $j$, respectivamente, interpretadas como vectores de reales, $M$ es el número de palabras del código y $d_{ij}$ es la distancia de Hamming entre las palabras $i$ y $j$. Obsérvese que, para palabras binarias, la distancia de Hamming coincide con el cuadrado de la distancia euclídea.

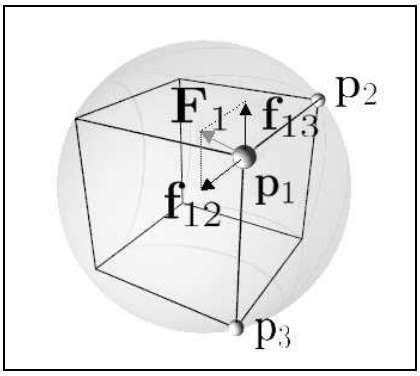


Fig. 3. Fuerzas entre tres partículas.

Una vez que conocemos la fuerza resultante sobre cada palabra, tenemos que simular el movimiento. Como se dijo anteriormente, las palabras sólo se pueden mover a vértices adyacentes. Esto significa invertir un bit de la palabra. Para averiguar sobre qué arista se moverá cada palabra, descomponemos el vector de fuerza resultante como suma de dos vectores perpendiculares. Uno de ellos normal al $n$-plano tangente a la $n$-esfera en que se haya circunscrito el $n$-cubo (componente normal), y el otro contenido en el $n$-plano tangente (componente tangencial). El primero lo denotaremos con $\mathbf{F}_i^n$ y el segundo con $\mathbf{F}_i^t$ (Fig. 4).

$$\mathbf{F}_i^n = (\mathbf{F}_i \cdot \hat{\mathbf{n}}_i)\hat{\mathbf{n}}_i \tag{3}$$

$$\mathbf{F}_i^t = \mathbf{F}_i - \mathbf{F}_i^n \tag{4}$$

donde $\hat{\mathbf{n}}_i$ es el vector unitario normal a la $n$-esfera que apunta hacia el exterior en $\mathbf{p}_i$, es decir:

$$\hat{\mathbf{n}}_i = \frac{\mathbf{p}_i - \frac{1}{2}\mathbf{o}_n}{|\mathbf{p}_i - \frac{1}{2}\mathbf{o}_n|}$$

donde $\mathbf{o}_n$ representa el vector con $n$ componentes todas a uno.

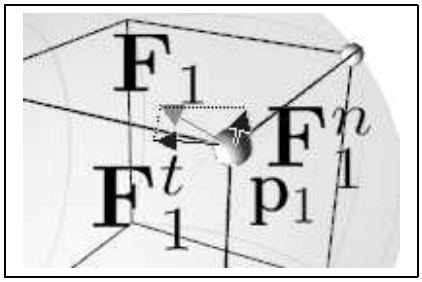


Fig. 4. Componentes normal y tangencial.

El vector perpendicular a la $n$-esfera se descarta porque contribuye al movimiento de la partícula lejos de la superficie de la $n$-esfera. Así que nos centramos en la componente tangencial ($\mathbf{F}^t_i$). Si las palabras se pudieran mover libremente por la superficie de la $n$-esfera, la componente tangencial representaría la aceleración del movimiento. Como las palabras pueden moverse solamente por una arista, necesitamos determinar la arista que forma el menor ángulo con la componente tangencial de la fuerza. Esta arista define el movimiento que conduce a un mayor descenso en la energía potencial. Para hacer esto, asignamos un vector unitario a cada arista. El origen del vector es el vértice en el que está la partícula, y el destino es el otro extremo de la arista. En cada vértice confluyen tantas aristas como dimensiones, así que denotaremos los vectores de aristas con $\mathbf{e}^k_i$, donde $i$ es la partícula y $k$ la dimensión (obsérvese que el par vértice-dimensión determina un vector de arista, pero para hacerlo más fácil, usamos $\mathbf{e}^k_i$ para referirnos al vector de arista asociado con el vértice $\mathbf{p}_i$). Después, hacemos los productos escalares $m^k_i = \mathbf{F}^t_i \cdot \mathbf{e}^k_i$ para cada partícula y dimensión, y elegimos para cada partícula $i$ la dimensión $k$ con mayor valor $m^k_i$ (mínimo ángulo). Esta dimensión determina la arista (bit) de movimiento, es decir, la palabra se moverá por esa arista (Fig. 5).

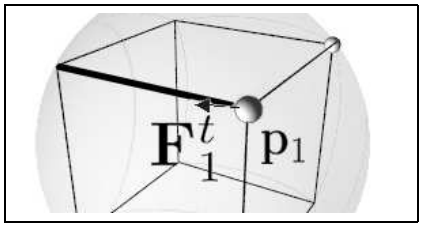


Fig. 5. Componente tangencial y arista de movimiento.

Si la componente tangencial de la fuerza es demasiado pequeña, la partícula está cerca de una posición de equilibrio. En la posición de equilibrio, la partícula no se puede mover porque ha alcanzado un mínimo local de la función de energía, asumiendo que las otras partículas están en reposo. Por esta razón, se exige un valor umbral $\tau$ en $m^k_i$ para realizar el movimiento. De todas las palabras susceptibles de ser movidas, es decir, aquéllas con $m^k_i \geq \tau$, se elige una aleatoriamente y se "mueve". Las otras partículas se mantienen intactas. Con este movimiento, se completa una iteración del algoritmo. La Fig. 6 muestra el pseudocódigo del algoritmo.

```
while not StopCriterion do
    no_move = ∅
    repeat
        if size(no_move) = M then
            exit;
        end if;
        repeat
            i=random (1,M);
        until not i in no_move;

        for j = 1 to M do
            if j ≠ i and d_ij ≠ 0 then
                f_ij = (p_i − p_j)/(d_ij √d_ij);
            end if;
        end for;
        F_i = Σ_{j=1,j≠i}^{M} f_ij;
        n̂_i = (p_i − ½o_n)/|p_i − ½o_n|;
        F^n_i = (F_i · n̂_i)n̂_i;
        F^t_i = F_i − F^n_i;
        max = 0;
        dim[i] = 0;
        for k = 1 to n then
            m = F^t_i · e^k_i;
            if m ≥ τ and m > max then
                max = m;
                dim[i] = k;
            end if;
        end for;
        if dim[i] = 0 then
            add(no_move, i)
        end if;
    until dim[i] >0;
    move (p_i, dim[i]);
end while;
```

Fig. 6. Pseudocódigo del Algoritmo de Repulsión (RA).

### C. Algoritmos Híbridos

En su sentido más amplio, la hibridación se refiere a la inclusión de conocimiento dependiente del problema en un algoritmo de búsqueda general [9]. Precisando más, podemos distinguir dos tipos de hibridación: *hibridación fuerte* e *hibridación débil* [10].

- **Hibridación Fuerte**: son algoritmos donde el conocimiento es incluido usando una representación u operadores específicos.
- **Hibridación Débil**: son algoritmos resultantes de la combinación de varios algoritmos.

Dentro de la hibridación débil, un algoritmo puede usarse para mejorar los resultados que ofrece otro, o aplicarse como operador del otro. En este trabajo hemos empleado dos algoritmos híbridos débiles que siguen este último esquema.

Así pues, el algoritmo ssGARA es similar al ssGA pero usa una iteración del algoritmo RA como operador de mutación. De igual forma, el algoritmo dGARA usa una iteración de RA como operador de mutación en el algoritmo dGA.

## IV. Estudio Experimental

En esta sección discutiremos la representación de las soluciones para el problema y la función de fitness empleada. Después, describiremos los experimentos y los parámetros usados. Finalmente, discutiremos los resultados obtenidos.

### *A. Representación y Función de Fitness*

Para codificar las soluciones del problema usamos una cadena binaria de longitud $M \times n$. El genotipo es la concatenación de las $M$ palabras de longitud $n$. Ésta es una representación directa de los parámetros del problema en el cromosoma de cada individuo, proporcionando una conversión sencilla entre el genotipo y el fenotipo.

Como función de fitness a maximizar, podría usarse la distancia mínima de Hamming entre las palabras del código, pero esta función es muy burda, ya que da poca información acerca de las ventajas internas de dos soluciones que compiten. Una función de fitness más precisa es la propuesta por Dontas y De Jong [3], es decir:

$$f(\mathbf{x}) = \frac{1}{\sum_{i=1}^{M}\sum_{j=1,j\neq i}^{M}\frac{1}{d_{ij}^2}} = \frac{1}{2\sum_{i=1}^{M-1}\sum_{j=i+1}^{M} d_{ij}^{-2}} \quad (5)$$

donde $\mathbf{x}$ es el código (vector de palabras) y $d_{ij}$ es la distancia de Hamming entre $x_i$ y $x_j$. Esta función mide cómo de bien están colocadas las $M$ palabras en las esquinas de un espacio $n$-dimensional considerando la mínima configuración de energía de $M$ partículas, tal y como se hace en Física. Aunque ha sido usada con éxito en el pasado ([2], [3], [4]), esta función tiene un inconveniente: puede asignar un valor más alto de fitness a un código con menor distancia mínima de Hamming que otro. Esto se puede ver en el siguiente ejemplo.

Sean $C_1$ y $C_2$ los dos códigos de diez bits y tres palabras que mostramos a continuación.

$$C_1 = \{0000000000, 0000011111, 0011100111\}$$

$$C_2 = \{0000000000, 0000001111, 1111110011\}$$

La distancia mínima en el primero es $d(C_1) = 5$, y en el segundo $d(C_2) = 4$. No obstante, sus valores de fitness de acuerdo con la Ecuación (5) son $f(C_1) = 4.639$ y $f(C_2) = 5.333$. Podemos observar que el código con mayor distancia mínima de Hamming entre palabras ($C_1$) tiene menor valor de fitness que el otro ($C_2$).

Para evitar este problema hemos añadido a la ecuación un término que aumenta con respecto a la distancia mínima de Hamming. La función de fitness que hemos usado es:

$$f'(\mathbf{x}) = \frac{1}{\sum_{i=1}^{M}\sum_{j=1,j\neq i}^{M}\frac{1}{d_{ij}^2}} + \left(\frac{d_{min}}{12} - \frac{d_{min}^2}{4} + \frac{d_{min}^3}{6}\right) \quad (6)$$

donde $d_{min}$ es la mínima distancia de Hamming entre palabras del código.

Esta función se descompone en dos términos. El primero es exactamente la Ecuación (5), que permite una distinción precisa entre códigos con la misma distancia mínima de Hamming pero diferente configuración de palabras. El segundo garantiza que los valores de fitness de códigos con distancia mínima $d$ sean menores que aquellos de códigos con distancia mínima $d+1$. Ahora explicaremos la expresión del segundo término.

En la Fig. 7 (izquierda) mostramos con barras verticales el rango de valores en el que se mueve la función de fitness para códigos con cierta distancia mínima usando la Ecuación (5). Podemos observar que hay solapamiento entre códigos con diferente distancia mínima. Ésta es la desventaja señalada anteriormente. En la Fig. 7 (derecha) mostramos la solución adoptada. El segundo término de la Ecuación (6) es la suma de las longitudes de los rangos de fitness asociados con las distancias mínimas previas. Así que, para derivar el segundo término necesitamos conocer las longitudes de los rangos de fitness. No es fácil calcular el valor exacto, pero afortunadamente es suficiente con una cota superior. Calculemos una cota superior.

Supongamos que tenemos un código con distancia mínima de Hamming $d$, una cota inferior del valor de la Ecuación (5) se obtiene cuando todos los $d_{ij}$ son $d$, es decir:

$$f_{ci} = \frac{1}{\sum_{i=1}^{M}\sum_{j=1,j\neq i}^{M}\frac{1}{d^2}} = \frac{d^2}{M(M-1)} \quad (7)$$

Ya que esta expresión depende de $M$, consideraremos 0 como cota inferior de $f$ para simplificar. Una cota superior de $f$ es alcanzada cuando todos los $d_{ij}$ son máximos. Asumiremos que tienden a infinito. No obstante, hay al menos dos $d_{ij}$ que son iguales a $d$, luego:

$$f_{cs} = \frac{1}{\frac{1}{d^2} + \frac{1}{d^2}} = \frac{d^2}{2} \quad (8)$$

Así que, una cota superior para la longitud del rango de fitness es $d^2/2$, y la función de fitness propuesta es:

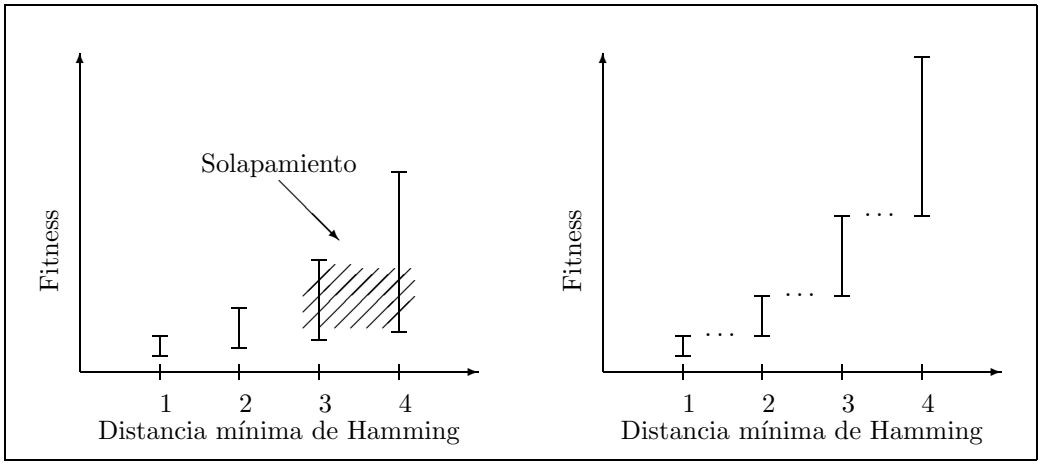


Fig. 7. Rango de fitness con la Ecuación (5) (izquierda) y la Ecuación (6) (derecha).

$$f'(\mathbf{x}) = \frac{1}{\sum_{i=1}^{M}\sum_{j=1,j\neq i}^{M}\frac{1}{d_{ij}^2}} + \sum_{i=1}^{d_{min}-1}\frac{i^2}{2} \quad (9)$$

en la cual, desarrollando el sumatorio, obtenemos la Ecuación (6).

### *B. Algoritmos y Parámetros*

Para resolver el problema usamos siete algoritmos: un algoritmo de repulsión (RA), un algoritmo genético de estado estacionario (ssGA), un algoritmo CHC, un algoritmo genético celular (cGA), un algoritmo genético distribuido (dGA), un híbrido entre ssGA y RA (ssGARA) y otro híbrido entre dGA y RA (dGARA). Como punto de partida en la hibridación con RA hemos optado por utilizar los GAs estándares (ssGA y dGA) y no los especializados (CHC y cGA). La hibridación con estos últimos queda como trabajo futuro. A continuación detallamos los parámetros de los algoritmos.

Para RA usamos como criterio de parada alcanzar $2 \cdot 10^5$ iteraciones o encontrar una solución óptima. El umbral $\tau$ se establece a 0.001.

Todos los GAs, a excepción de CHC, usan selección por torneo binario, recombinación de un punto con probabilidad $p_c = 1.0$, mutación por inversión de bits con probabilidad $p_m = 0.003$ (aproximadamente la inversa de la longitud de la cadena) de invertir un bit y reemplazo elitista. CHC difiere de lo anterior en el operador de recombinación ($HUX$) y en el de selección (propia de CHC).

Para ssGA usamos una población de 480 individuos y $10^5$ iteraciones como máximo (200480 evaluaciones). La configuración base utilizada en ssGA (pocas iteraciones y un población relativamente grande) no es adecuada para el CHC. Este algoritmo suele necesitar un número de generaciones mayor pero a cambio trabaja muy bien con poblaciones más pequeñas. Con la configuración base, tanto CHC como cGA obtienen unos resultados iniciales muy pobres, por lo que decidimos probar con diferentes configuraciones de población e iteraciones, manteniendo constante el número de evaluaciones para que el esfuerzo total resultante fuera equivalente en todos los algoritmos. Las configuraciones con las que se han obtenido mejores resultados se describen a continuación. Para el CHC usamos una población de 30 individuos, un máximo de 6720 iteraciones. Para el cGA usamos una población 100 individuos con un máximo de 1000 iteraciones. La Tabla I resume los parámetros de los algoritmos anteriores.

| | CHC | ssGA | cGA |
|---|---|---|---|
| Población | 30 | 480 | 100 |
| Selección | | Torneo Bin. (2 ind.) | |
| Recomb. | HUX | SPX ($p_c = 1.0$) | |
| Mutación | Bit-Flip ($p_m = 0.003$) | | |
| Reemplazo | Elitista | | |
| Máx. iters. | 6720 | $10^5$ | 1000 |

| | RA |
|---|---|
| $\tau$ | 0.001 |
| Máx. iters. | $2 \cdot 10^5$ |

TABLA I
PARÁMETROS DE LOS ALGORITMOS CANÓNICOS.

Usamos tres dGAs con 5, 10 y 15 islas para resolver el problema. Los llamamos dGA5, dGA10 y dGA15, respectivamente. El tamaño de la población total es el mismo en los tres dGAs, es decir, 480 individuos. Estos individuos se distribuyen equitativamente entre las islas. Los subalgoritmos están conectados siguiendo una topología en anillo unidireccional con migración asíncrona. Cada once iteraciones se elige un individuo de la población por torneo binario y se envía al si-

guiente subalgoritmo del anillo. En el destino, el individuo es insertado en la población si es mejor que el peor de la población. El criterio de parada es encontrar un óptimo o alcanzar $10^5$ iteraciones entre todos los subalgoritmos (200480 evaluaciones). Los parámetros de los algoritmos se resumen en la Tabla II.

| | dGAxx5 | dGAxx10 | dGAxx15 |
|---|---|---|---|
| Islas | 5 | 10 | 15 |
| Tam. Subpob. | 96 | 48 | 32 |
| Selección | Torneo Bin. (2 inds.) | | |
| Recombinación | SPX ($p_c = 1.0$) | | |
| Explotación | dGA: Bit-Flip ($p_m = 0.003$) dGARA: 1 iter. RA ($\tau = 0.001$) | | |
| Reemplazo | Elitista | | |
| Topología | Anillo Unidir. | | |
| Tipo Migración | Asíncrona | | |
| Periodo Migr. | 11 | | |
| Selec. Migr. | Torneo Bin. (1 ind.) | | |
| Reemp. Migr. | Peor local si entrante mejor | | |
| Máx. iters. | $10^5$ | | |

TABLA II
PARÁMETROS DE LOS DGA$n$ Y DGARA$n$.

Los parámetros para ssGARA son los mismos que los de ssGA y RA por separado. El algoritmo dGARA es similar a dGA, pero usa RA como operador de mutación, tal y como hace ssGARA. Usamos tres algoritmos dGARA con 5, 10 y 15 islas: dGARA5, dGARA10 y dGARA15. Los parámetros están resumidos en la Tabla II.

Todos los algoritmos fueron ejecutados en una sola CPU. Las máquinas usadas para los experimentos son Pentium 4 a 2.4GHz con 512MB de RAM. Presentamos los valores medios de 30 ejecuciones independientes.

En los algoritmos distribuidos, el número de evaluaciones necesarias para alcanzar un óptimo es la suma de todas las evaluaciones hechas por los subalgoritmos.

### *C. Experimentos y Resultados*

En esta subsección discutiremos el rendimiento relativo de los algoritmos presentados. Comenzaremos con los algoritmos canónicos (Tabla III).

| | % | Mejor fitness | | Evaluaciones | |
|---|---|---|---|---|---|
| | Éxito | $\overline{x}$ | $\sigma_n$ | $\overline{x}$ | $\sigma_n$ |
| RA | 0.00 | 3.16 | 1.53 | – | – |
| ssGA | 0.00 | 7.06 | 0.00 | – | – |
| CHC | 6.67 | 8.43 | 5.11 | 135465.00 | 27885.00 |
| cGA | 3.33 | 7.75 | 3.68 | 94300.00 | 0.00 |

TABLA III
RESULTADOS DE RA, SSGA, CHC Y CGA PARA ECC.

En esta tabla se observa que CHC es el que mejor media de fitness obtiene, y junto al cGA son los únicos algoritmos que encuentran una solución óptima, aunque muy infrecuentemente. Además, en ambos algoritmos se observó que en muchas ejecuciones la diversidad final de la población era relativamente alta, lo que nos hizo pensar que con un número mayor de evaluaciones el algoritmo sería capaz de mejorar los resultados de la tabla. Aumentando el número máximo de evaluaciones, logramos aumentar de forma notable los resultados anteriores, obteniendo un 40 % de éxito con 862588 evaluaciones de media en el caso del CHC y un 20 % de éxito con 622259 evaluaciones de media y 484 individuos en la población con el cGA. Se puede apreciar que ni RA ni ssGA son capaces de encontrar una solución óptima en ningún caso. No obstante, el mejor fitness medio obtenido por ssGA es más alto que el de RA, así que podemos decir que las soluciones de ssGA están más cercanas al óptimo, en general, que las de RA.

En la Tabla IV presentamos los resultados de los dGAs. Vemos que tan sólo dGA10 es capaz de encontrar la solución en 1 de las 30 ejecuciones. Al igual que ocurre con CHC y cGA, un aumento en el número de evaluaciones mejora los resultados. En ese caso, dGA5 consigue un 13.33 % de éxito con 265179 evaluaciones medias, mientras que dGA10 y dGA15 consiguen un 20 % con 656393 y 2144207 evaluaciones medias, respectivamente.

| dGA$n$ | | | | | |
|---|---|---|---|---|---|
| | % | Mejor fitness | | Evaluaciones | |
| $n$ | Éxito | $\overline{x}$ | $\sigma_n$ | $\overline{x}$ | $\sigma_n$ |
| 15 | 0.00 | 7.06 | 0.00 | – | – |
| 10 | 3.33 | 7.75 | 3.68 | 198310.00 | 0.00 |
| 5 | 0.00 | 7.06 | 0.00 | – | – |

TABLA IV
RESULTADOS DE LOS DGA$n$ PARA ECC.

A continuación, en la Tabla V, presentamos los resultados de los dos algoritmos híbridos utilizados: ssGARA y dGARA. Para el segundo, tal y como hicimos con los dGAs, hemos empleado tres variantes que cambian en el número de islas.

| ssGARA | | | | | |
|---|---|---|---|---|---|
| | % | Mejor fitness | | Evaluaciones | |
| | Éxito | $\overline{x}$ | $\sigma_n$ | $\overline{x}$ | $\sigma_n$ |
| | 53.33 | 19.07 | 9.42 | 65291.00 | 35793.92 |
| $n$ | dGARA$n$ | | | | |
| 15 | 90.00 | 25.78 | 5.48 | 72722.38 | 29155.48 |
| 10 | 90.00 | 26.05 | 4.76 | 69526.44 | 23997.12 |
| 5 | 83.33 | 24.68 | 6.64 | 63265.76 | 21696.38 |

TABLA V
RESULTADOS DE SSGARA Y DGARA$n$ PARA ECC.

Lo primero que observamos es que las versiones híbridas de los algoritmos superan en todos los casos el porcentaje de éxito de las versiones puras tanto de GAs como de RA, alcanzando 90.00 % en dGARA10 y dGARA15. Además, los números de evaluaciones empleados para encontrar las

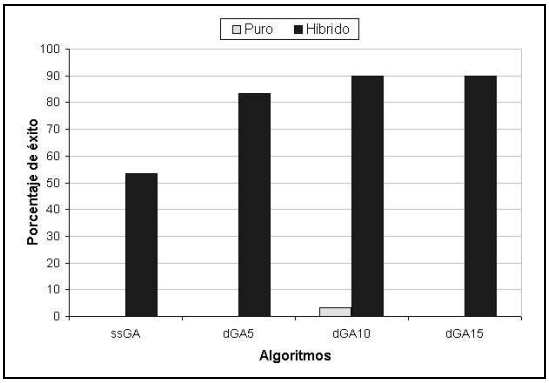


Fig. 8. Algoritmos puros frente a híbridos.

soluciones son menores que para todos los algoritmos previos.

En la Fig. 8 mostramos en un diagrama de barras, el porcentaje de éxito de los algoritmos ssGA, dGA5, dGA10, dGA15 y sus versiones híbridas. Se observa una clara ventaja de la hibridación con RA.

Todos los resultados anteriores son mejores que otros presentados en la literatura. En [2] se usa una arquitectura SIMD y el algoritmo GSA para resolver el problema. El algoritmo requiere más de 12000 iteraciones paralelas con 256 procesadores (más de 3072000 evaluaciones) y más de 10000 con 16284 procesadores (más de 163840000 evaluaciones) para alcanzar una solución óptima. En [4] encontramos resultados competitivos con los del presente trabajo, pero la representación usada para el problema es diferente y no puede generalizarse a la clase ECC, sino que sólo es válida para ciertas instancias ($n = 12$, $M = 24$ entre ellas).

## V. Conclusiones y Trabajo Futuro

En este trabajo hemos aplicado algoritmos genéticos en panmixia y estructurados, así como un nuevo algoritmo de búsqueda local y varios algoritmos híbridos para resolver el problema ECC. Las conclusiones obtenidas son que la descentralización y, muy especialmente, la hibridación con RA son dos técnicas que favorecen la búsqueda de un óptimo del problema. El uso de ambas técnicas simultáneamente (algoritmos distribuidos e híbridos) consigue mejores resultados que su actuación por separado.

Como trabajo futuro, estudiaremos otras hibridaciones como CHC y RA o cGA y RA. Aplicaremos las técnicas híbridas distribuidas a otros problemas, en particular, podemos emplear híbridos con RA a problemas donde pueda establecerse la analogía con la Física, como ocurre con el problema de Thomson [11] que consiste en colocar cargas en la superficie de una esfera para llegar a un estado de equilibrio.

## Agradecimientos

Este trabajo ha sido subvencionado parcialmente por el MCyT y FEDER bajo el contrato TIC2002-04498-C05-02 (el proyecto TRACER).

## Referencias

[1] S. Lin and D.J. Costello, *Error Control Coding: Fundamentals and Applications*, Prentice Hall NJ, 1983.

[2] H. Chen, N. S. Flann, and D. W. Watson, "Parallel Genetic Simulated Annealing: A Massively Parallel SIMD Algorithm," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 2, pp. 126–136, 1998.

[3] K. Dontas and K. De Jong, "Discovery of maximal Distance Codes Using Genetic Algorithms," in *Proceedings of the 2nd International IEEE Conference on Tools for Artificial Intelligence*, Herndon, VA, 1990, pp. 905–811, IEEE Computer Society Press, Los Alamitos, CA.

[4] E. Alba, C. Cotta, F. Chicano, and A. J. Nebro, "Parallel Evolutionary Algorithms in Telecommunications: Two Case Studies," in *Proceedings of the CACIC02*, 2002.

[5] E. Agrell, A. Vardy, and K. Zeger, "A Table of Upper Bounds for Binary Codes," *IEEE Transactions on Information Theory*, vol. 47, no. 7, pp. 3004–3006, 2001.

[6] T. Bäck, D. B. Fogel, and Z. Michalewicz, *Handbook of Evolutionary Computation*, Oxford University Press, New York NY, 1997.

[7] L. J. Eshelman, "The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination," in *Foundations of Genetic Algorithms*. 1991, pp. 265–283, Morgan Kaufmann.

[8] V. S. Gordon and D. Whitley, "Serial and Parallel Genetic Algorithms as Function Optimizers," in *Proceedings of the Fifth International Conference on Genetic Algorithms*, S. Forrest, Ed. 1993, pp. 177–183, Morgan Kaufmann.

[9] L. Davis, Ed., *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.

[10] C. Cotta and J. M. Troya, "On decision-making in strong hybrid evolutionary algorithm," *Tasks and Methods in Applied Artificial Intelligence*, vol. 1416, pp. 418–427, 1998.

[11] J.R. Morris, D.M. Deaven, and K.M. Ho, "Genetic-algorithm energy minimization for point charges on a sphere," *Physical Review B*, vol. 53, no. 4, pp. 1740–1743, 1996.