ICT-UNPA-127-2015
ISSN: 1852-4516
Aprobado por Resolución N° 1071/15-R-UNPA

Algoritmos de Búsqueda Dispersa aplicados a problemas de Optimización Discreta

Soria C[1]., Pandolfi D[2]., Villagra S[2]., Villagra A[2].
[1]{BecariodeInvestigaciondeGrado}
[2]{DocenteInvestigadorUNPA}
christianlsoria@hotmail.com, {dpandolfi, svillagra, avillagra@uaco.unpa.edu.ar}

UNPA UACO
Universidad Nacional de la Patagonia Austral - Unidad Académica Caleta Olivia
Departamento de Ciencias Exactas y Naturales
LabTEm- Laboratorio de Tecnologías Emergentes
Caleta Olivia, 2014

Resumen: La obtención de soluciones óptimas para muchos problemas de optimización, en el campo científico e industrial es intratable. Esto significa que un método exacto necesita un tiempo polinomial para garantizar la optimalidad de la solución. Esta clase de problemas denominados NP-duros, requieren de métodos que garanticen soluciones de alta calidad en un tiempo razonable aunque no garantice encontrar una solución óptima global. A éstos últimos se los denomina métodos aproximados o heurísticos, y dentro de ellos encontramos a las metaheurísticas. La Búsqueda Dispersa es una metaheurística que pertenece a la familia de los llamados Algoritmos evolutivos, los cuales se distinguen por estar basados en la combinación de un conjunto de soluciones. Si bien fue originalmente introducido a fines de los setenta, recientemente es cuando ha sido utilizado en numerosos problemas con gran éxito. La Búsqueda Dispersa proporciona un marco flexible que permite el desarrollo de diferentes implementaciones con distintos grados de complejidad. El objetivo del trabajo es presentar y comparar dos versiones de algoritmos de búsqueda dispersa aplicando un completo análisis estadístico. Se pretende estudiar el comportamiento de estos algoritmos en la solución de un conjunto de problemas de optimización. De los resultados obtenidos se determina que la segunda versión propuesta es la más adecuada para resolver el conjunto de problemas planteados.

Palabras clave: Búsqueda Dispersa, problemas de optimización, metaheurísticas.

1. INTRODUCCIÓN

La optimización en un contexto científico es el proceso de tratar de encontrar la mejor solución posible para un problema determinado. Cuando hablamos de problemas de optimización, estamos haciendo referencia a problemáticas reales generalmente de difícil solución, las cuales se encuentran en distintas áreas de aplicación, presentando una gran complejidad computacional para ser resueltos (NP-duros). Un problema de este tipo es aquel para el que no podemos garantizar el encontrar la mejor solución posible en un tiempo razonable. La resolución de los mismos se realiza a través de algoritmos aproximados. Estos proporcionan soluciones consideradas buenas para un determinado problema, en un tiempo moderado, pero no necesariamente la óptima. Estos métodos, en los que la rapidez del proceso es tan importante como la calidad de la solución obtenida, se denominan heurísticos o aproximados. Con el propósito de obtener mejores resultados que los alcanzados por los heurísticos tradicionales surgen los denominados procedimientos metaheurísticos. Estos procedimientos son una clase de métodos aproximados que están diseñados para resolver problemas de optimización complejos, en los que los heurísticos clásicos no son efectivos. Las metaheurísticas proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos (Martí 2003, Osman y Kelly 1996). La aplicación de técnicas metaheurísticas es especialmente interesante en caso de problemas de optimización combinatoria: problemas donde las variables de decisión son enteras (o discretas, al menos) y, generalmente, el espacio de soluciones está formado por ordenaciones de valores de dichas variables. Sin embargo, las técnicas metaheurísticas se pueden aplicar también a problemas de otro tipo, como con variables continuas, por ejemplo. Las técnicas metaheurísticas más extendidas son las siguientes: los Algoritmos Genéticos (GA) (Holland 1992), la Búsqueda Tabú (*Tabu Search*) (Glover 1989), el Recocido Simulado (*Simulated Annealing*) (Van Laarhoven y Aarts 1987), la Búsqueda Dispersa (*Scatter Search*) (Glover et al. 2000), las Colonias de Hormigas (ACO), entre otras.

En este trabajo se presentan dos propuestas de Búsqueda Dispersa (BD), mostrando las principales características de cada una de ellas y analizando resultados obtenidos a determinados problemas de optimización. La Búsqueda Dispersa, es un procedimiento metaheurístico basado en formulaciones y estrategias introducidas en la década del sesenta. Esta metaheurística opera sobre un conjunto de soluciones, llamado conjunto de referencia, combinando éstas para crear nuevas soluciones de modo que mejoren a las que las originaron. En este sentido decimos que es un método evolutivo. Sin embargo, a diferencia de otros métodos evolutivos, como los algoritmos genéticos, BD no está fundamentado en la aleatorización sobre un conjunto relativamente grande de soluciones sino en elecciones sistemáticas y estratégicas sobre un conjunto pequeño. Una de las características más notables de la Búsqueda Dispersa es que se basa en integrar la combinación de soluciones con la Búsqueda Local, aunque en diseños avanzados esta Búsqueda Local, no necesariamente, puede contener una estructura de memoria (*Tabu Search*). La primera descripción del método fue publicada por Glover (1977), donde establece los principios de la BD. En este primer artículo se determina que la BD realiza una exploración sistemática sobre una serie de *n* buenas soluciones llamadas conjunto de referencia. En este trabajo se proponen dos versiones que utilizan mecanismos diferentes para generar el conjunto de soluciones más dispersas. Además se realiza el estudio comparativo entre las dos versiones del método BD, aplicadas a problemas de optimización discretos, con el fin de determinar con cuál de las dos versiones se

encuentran mejores resultados, y si existen diferencias estadísticamente significativas entre una y otra, para los problemas utilizados.

El trabajo está organizado de la siguiente manera. En la Sección 2 se presenta una introducción a las metaheurísticas. En la Sección 3 se describe conceptualmente un Algoritmo de Búsqueda Discreta. En la Sección 4 se expone las dos versiones de Búsqueda Dispersa propuestas en este trabajo. En la Sección 5 se presentan los problemas de optimización combinatoria. En la Sección 6 se describe el diseño de los experimentos y algunos resultados obtenidos de los mismos. Finalmente, en la Sección 7, se exponen las conclusiones arribadas y líneas de trabajos a futuro.

2. METAHEURÍSTICAS

Con el fin de encontrar soluciones de buena calidad a problemas de optimización combinatoria, durante la década del setenta se han diseñado diversos métodos, conocidos como heurísticos. La mayor parte de estos métodos partieron de problemas de fácil representación, pero de muy difícil solución. Debido a la variada naturaleza de estos problemas, los métodos eran útiles apenas para el problema en el cual habían sido creados.
En los años ochenta surgen las técnicas metaheurísticas, las cuales plantean un cambio importante en el desarrollo de técnicas alternativas. La idea básica de las metaheurísticas era combinar diferentes métodos heurísticos a un nivel más alto, para conseguir una exploración del espacio de búsqueda de forma eficiente y efectiva. El término metaheurística fue introducido por primera vez por Glover (1986). Antes de que el término fuese aceptado por la comunidad científica estas técnicas eran conocidas como heurísticas modernas (Reeves 1993).

Todas las técnicas metaheurísticas tienen las siguientes características (Sait y Youssef 1999):

- Son ciegas, no saben si llegan a la solución óptima. Por lo tanto, se les debe indicar cuándo deben detenerse.
- Son algoritmos aproximativos y, por lo tanto, no garantizan la obtención de la solución óptima.
- Aceptan ocasionalmente malos movimientos (es decir, se trata de procesos de búsqueda en los que cada nueva solución no es necesariamente mejor en términos de la función objetivo que la inmediatamente anterior). Algunas veces aceptan, incluso, soluciones no factibles como paso intermedio para acceder a nuevas regiones no exploradas.
- Son relativamente sencillos; todo lo que se necesita es una representación adecuada del espacio de soluciones, una solución inicial (o un conjunto de ellas) y un mecanismo para explorar el campo de soluciones.
- Son generales. Prácticamente se pueden aplicar en la resolución de cualquier problema de optimización de carácter combinatorio. Sin embargo, la definición de la técnica será más o menos eficiente en la medida en que las operaciones tengan relación con el problema considerado.
- La regla de selección depende del instante del proceso y de la historia hasta ese momento. Si en dos iteraciones determinadas, la solución es la misma, la nueva solución de la siguiente iteración no tiene por qué ser necesariamente la misma. En general, no lo será.

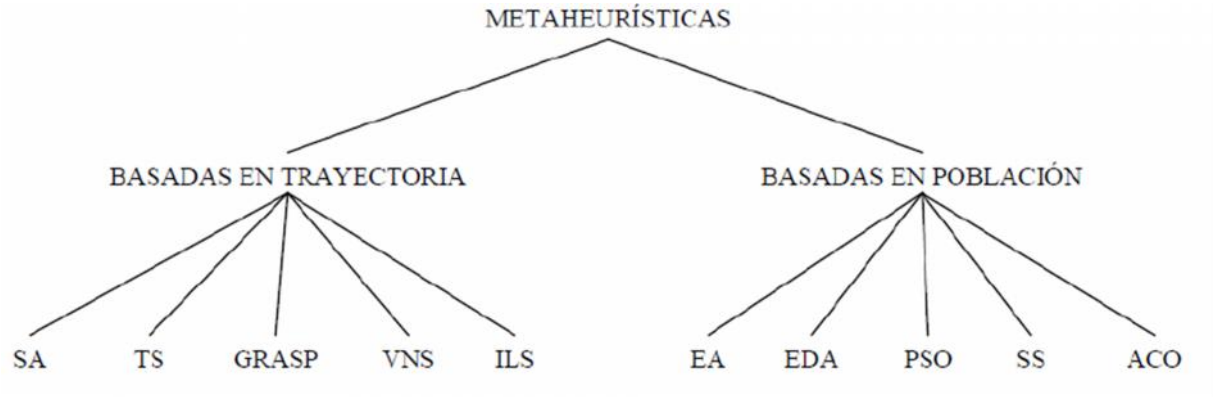

Figura 1: Clasificación de las metaheurísticas

Según sus características, las metaheurísticas se pueden clasificar de distintas maneras. Es posible clasificarlas en metaheurísticas inspiradas en la naturaleza versus las no inspiradas, basadas en memoria o sin ella, o en métodos que usen funciones objetivo estáticas o dinámicas, etc. Hoy en día una de las clasificaciones más populares, las describen teniendo en cuenta si utilizan en cada paso un único elemento del espacio de búsqueda, denominadas metaheurísticas basadas en trayectoria, o si trabajan con un conjunto de puntos o población, en este caso referidas como metaheurísticas basadas en población. Esta taxonomía se muestra de forma gráfica en la Figura 1, la cual incluye las técnicas más representativas. Las siglas de la figura se corresponden de la siguiente manera: *Simulated Annealing* (SA), *Tabu Search* (TS), *Greedy Randomized Adaptive Search Procedure* (GRASP), *Variable Neighborhood Search* (VNS), *Iterated Local Search* (ILS), *Evolutionary Algorithms* (EA), *Estimation Distribution Algorithm* (EDA), *Particle Swarm Optimization* (PSO), *Scatter Search* (SS), y *Ant Colony Optimization* (ACO).

La principal característica de las metaheurísticas basadas en trayectoria, es que parten de una solución y, mediante la exploración del vecindario, van actualizando la solución actual, formando una trayectoria. La mayoría de estos algoritmos surgen como extensiones de los métodos simples de Búsqueda Local a los que se les añade algún mecanismo para escapar de los mínimos locales. Normalmente se termina la búsqueda cuando se alcanza un número máximo predefinido de iteraciones, se encuentra una solución con una calidad aceptable, o se detecta un estancamiento del proceso. Mientras que las basadas en población se caracterizan por trabajar con un conjunto de soluciones, usualmente denominado población en cada iteración, los algoritmos basados en población proveen una forma natural e intrínseca de explorar el espacio de búsqueda, a diferencia de los métodos basados en trayectoria, que únicamente manipulan una solución del espacio de búsqueda por iteración.

En la próxima sección se desarrolla el método BD, propuesto para la realización de este trabajo, el cual es una metaheurística basada en la población.

3. ALGORITMO DE BÚSQUEDA DISPERSA

La Búsqueda Dispersa es un método evolutivo que ha sido aplicado en la resolución de un gran número de problemas de optimización (Alba y Luque 2003, Gendreau y Potvin 2010, Laguna et al. 20014, Li y Tian 2015). Los conceptos y principios fundamentales del método, fueron propuestos al comienzo de la década de los setenta por Glover (1977), basados en las

estrategias para combinar reglas de decisión, especialmente en problemas de secuenciación, así como en la combinación de restricciones. En Glover (1994) se introduce la combinación ponderada (*weighted combination*) como el mecanismo principal para generar nuevas soluciones. En esta versión se enfatizan las búsquedas lineales entre dos soluciones y el uso de pesos para muestrear en dicha línea. Asimismo, se introduce el concepto de combinar soluciones de calidad con soluciones diversas (soluciones que no son consideradas de calidad, seleccionadas aleatoriamente). Además, el método incluye una componente de intensificación que consiste en tomar una muestra mayor de la línea que ha producido mejores soluciones. En Glover (1998) se publica una versión más específica del método en donde se recogen y simplifican muchas de las ideas expuestas en trabajos anteriores. Finalmente, Glover, Laguna y Martí (2000) estudian las implementaciones más recientes del método en la resolución de problemas de optimización combinatoria (2003). La Búsqueda Dispersa se basa en combinar las soluciones que aparecen en el llamado conjunto de referencia. Este conjunto almacena las "buenas" soluciones que se han ido encontrando durante el proceso de búsqueda. Es importante destacar que el significado de buena no se restringe a la calidad de la solución, sino que también se considera la diversidad que esta aporta al conjunto de referencia. BD consta básicamente de cinco elementos o métodos que se describen a continuación:

1. **Generador de Soluciones Diversas**. El método se basa en generar un conjunto *P* de soluciones diversas (diversas con respecto a las buenas soluciones), del que extraeremos un subconjunto pequeño que denominamos conjunto de referencia *RefSet*.

2. **Método de Mejora.** Típicamente se trata de un método de Búsqueda Local para mejorar las soluciones, tanto del conjunto de referencia como las combinadas antes de estudiar su inclusión en el conjunto de referencia. Es importante destacar que en las implementaciones donde se manejen soluciones no factibles, este método es capaz de, a partir de una solución no factible, obtener una que sea factible y después intentar mejorar su valor. Si el método no logra mejorar a la solución inicial, se considera que el resultado es la propia solución inicial.

3. **Método para crear y actualizar el conjunto de referencia** *RefSet*. A partir del conjunto de soluciones diversas *P* se extrae el conjunto de referencia según el criterio de contener soluciones de calidad y diferentes entre sí (Calidad y Diversidad). Las soluciones en este conjunto están ordenadas de mejor a peor respecto de su calidad.

a. **Creación**. Iniciamos el conjunto de referencia con las *b/2(|RefSet|=b) mejores* soluciones de *P* (subconjunto R_1). Las *b/2* restantes se extraen de *P* con el criterio de maximizar la mínima distancia con las ya incluidas en el conjunto de referencia (subconjunto R_2). Para ello debemos de definir previamente una función de distancia en el problema.
b. **Actualización.** Las soluciones producto de las combinaciones pueden entrar en el conjunto de referencia y reemplazar a alguna de las ya incluidas, en caso de que las mejoren. Así, el conjunto de referencia mantiene un tamaño *b* constante, pero el valor de sus soluciones va mejorando a lo largo de la búsqueda. En implementaciones sencillas, la actualización de este conjunto se realiza únicamente por calidad, aunque se puede hacer también por diversidad.

4. **Método para generar subconjuntos de** *RefSet*. BD se basa en examinar de una forma bastante exhaustiva todas las combinaciones del *RefSet*. Este método especifica la forma en que se seleccionan los subconjuntos para aplicarles el método de combinación. Una implementación sencilla, utilizada a menudo, consiste en restringir la búsqueda a parejas de soluciones. Así el método considera todas las parejas que se pueden formar con los elementos

del *RefSet* y a todas ellas le aplica el método de combinación.

5. **Método de combinación**. BD se basa en combinar todas las soluciones del conjunto de referencia. Para esto, se consideran los subconjuntos formados por el método del paso 4, y se les aplica el método de combinación. La solución o soluciones que se obtienen de esta combinación pueden ser inmediatamente introducidas en el conjunto de referencia (actualización dinámica) o almacenadas temporalmente en una lista hasta terminar de realizar todas las combinaciones y después ver qué soluciones entran en éste (actualización estática).
En el siguiente esquema (Matrí y Laguna 2003) se muestra cómo actúan los elementos descritos en un esquema básico del algoritmo. Este hace referencia a los subconjuntos de *R* ya que podemos combinar parejas, tríos o cualquier número de soluciones. Es usual limitar las combinaciones a parejas, por lo que el punto 6 equivaldría a decir: "Generar todas las parejas de soluciones de *R* en las que al menos una de las dos sea nueva"; donde por nueva entenderemos que haya entrado al conjunto después de realizar la última combinación de todo *RefSet*.

***Algoritmo** Búsqueda Dispersa*

1. *Comenzar con $P = \varnothing$. Utilizar el **método de generación** para construir una solución **método de mejora** para tratar de mejorarla; sea x la solución obtenida. Si $x \notin P$ entonces añadir x a $P.(i.e., P = P \cup x)$, en otro caso, rechazar x. Repetir esta etapa hasta que P tenga un tamaño prefijado.*
2. *Construir el **conjunto de referencia** $RefSet = x^1, \ldots, x^b$ con las $b/2$ mejores soluciones de P (subconjunto R_1) y las $b/2$ soluciones de P más diversas a las ya incluidas (subconjunto R_2).*
3. *Evaluar las soluciones en $RefSet$ y ordenarlas de mejor a peor respecto a la función objetivo.*
4. *Hacer $NuevaSolución = TRUE$*

   ***Mientras** (NuevaSolución)*
5. *$NuevaSolucion = FALSE$*
6. *Generar los subconjuntos de $RefSet$ en los que haya al menos una nueva solución.*

   ***Mientras** (Queden subconjuntos sin examinar)*
   7. *Seleccionar un subconjunto y etiquetarlo como examinado.*
   8. *Aplicar el **método de combinación** a las soluciones del subconjunto.*
   9. *Aplicar el **método de mejora** a cada solución obtenida por combinación. Sea x la solución mejorada: Si$(f(x) < f(x^b)$ y x no está en $RefSet)$.*
   10. *Hacer $x^b = x$ y reordenar $Refset$*
   11. *Hacer $NuevaSolucion = TRUE$*

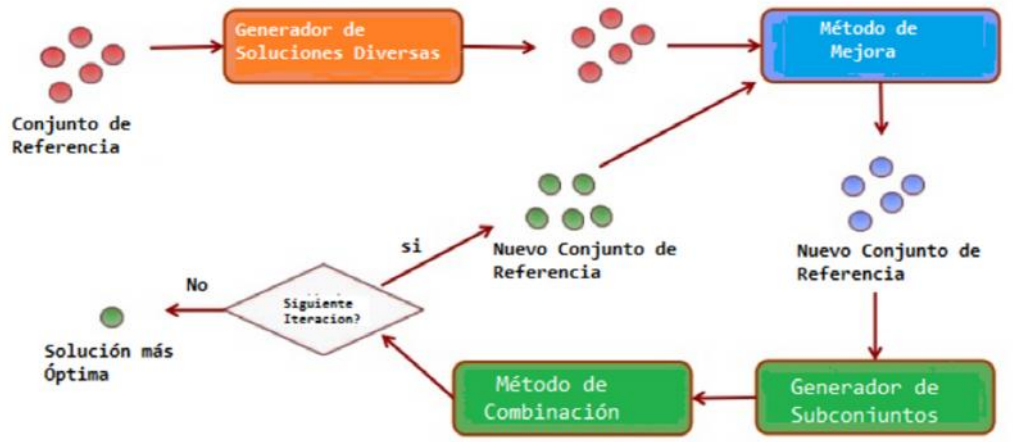

Figura 2: Esquema Básico Algoritmo de Búsqueda Dispersa.

Se puede notar que el algoritmo se detiene cuando al tratar de combinar se ve que no hay nuevos elementos en el conjunto de referencia (la variable NuevaSolución tiene valor FALSE). Este algoritmo puede ser anidado en un esquema global que permita reconstruir el conjunto de referencia cuando éste ya ha sido utilizado. Así, si el límite de tiempo (o evaluaciones) no se ha excedido, una estrategia es regenerar el conjunto de referencia, dejando la mitad superior (*b/2* mejores) y eliminando la mitad inferior. Después, se genera un conjunto *P* como al comienzo del algoritmo, del que se extraen únicamente las *b/2* soluciones más diversas con las ya existentes en *R*. De esta forma obtenemos un nuevo conjunto de referencia en el que mantenemos las soluciones de calidad y renovamos las correspondidas a diversidad. Se vuelve a combinar como anteriormente sobre este conjunto de referencia (pasos 5 a 11). De este modo se obtiene un esquema cíclico indefinido al que hay que añadirle una variable de control para detenerlo. Habitualmente esta variable está en función del tiempo o del número de iteraciones (evaluaciones de la función objetivo). En la Figura 2 se muestra el esquema básico de un algoritmo de Búsqueda Dispersa, donde se pueden observar los métodos básicos y la interacción de los mismos.

En la siguiente sección presentaremos las dos versiones de Búsqueda Dispersa propuestas para este trabajo.

4. ALGORITMOS PROPUESTOS

En esta sección se describen las versiones de Búsqueda Dispersa que estamos proponiendo y que se diferencian en la forma en que se genera el subconjunto de las soluciones más diversas. A la primera propuesta se la denomina BDG que significa “Búsqueda Dispersa Global” y la segunda propuesta la denominamos BDE “Búsqueda Dispersa Estratificada”.

4.1 BÚSQUEDA DISPERSA GLOBAL

El funcionamiento de esta versión (BDG) se basa en la explicación general del algoritmo de Búsqueda Dispersa. La variante que se muestra aquí es la forma en cómo se genera el subconjunto R_1 de las soluciones más diversas. Se denomina BDG, porque toma de manera global todas las soluciones, las ordena en cuanto a su calidad y selecciona las más diversas. Se

toma la diversidad en base a la calidad, es decir, que las soluciones más diversas son las de peor calidad. El funcionamiento de todo el algoritmo es el siguiente: Se genera la población inicial de tamaño *N(N = 100)* de soluciones de forma aleatoria. Se mejoran esas soluciones utilizando un algoritmo de búsqueda local. En este caso se aplica *Hill Climbing* (HC). A continuación se ordena la población utilizando un método de ordenamiento (*QuickSort*). Se toman las n_1 *(*n_1 *= 8)* mejores soluciones para formar el subconjunto R_1 y las n_2 *(*n_2 *= 8)* peores soluciones para formar el subconjunto R_2. Estos pasos se repiten mientras no se encuentre la condición de fin y se hayan generado nuevas soluciones, recombinando todas las soluciones del conjunto *R*. A las soluciones hijas generadas de esa recombinación (método de recombinación Dpx, un operador de recombinación que genera un solo individuo que su parte mayor está formado por el mejor padre). Se mejora utilizando un método de búsqueda local (en esta caso se utilizó HC), se ordenan las soluciones hijas con un método de ordenamiento (*QuickSort*). Se seleccionan las mejores soluciones y se remplazan las del subconjunto R_1 si éstas son mejores en cuanto a su calidad. Se seleccionan las más diversas (en este caso las peores) y se reemplazan a las del subconjunto R_2 si las superan en diversidad. Si no se ha alcanzado la condición de fin y no hay mejoras, es decir que el subconjunto R_1 no ha sido reemplazado por ninguna nueva solución, se regenera el subconjunto R_2, de la siguiente manera: se genera una nueva población de soluciones aleatorias de tamaño *N(N = 100)*, luego se ordenan esas soluciones y las n_2 más diversas (en este caso las peores en calidad) reemplazan a todas las soluciones del subconjunto R_2. En el Algoritmo 1 se muestra el pseudocódigo de la BDG.

Algoritmo 1 BDG

```
1: GeneraConjuntoP(P)
2: P' = OrdenarConjunto(P)
3: P'' = Mejora(P')
4: R = GeneraConjuntoR(P'')
5: while criteriodeFinalizacionnoAlcanzado do
6:    S = Combinacion(R)
7:    S' = MejoraSoluciones(S)
8:    S'' = OrdenaSoluciones(S')
9:    Actualizacion(R, S'')
10:   if NoNuevasSoluciones then
11:      EntoncesRegenerar(R)
12:   end if
13: end while
```

A continuación se muestra un esquema del funcionamiento general de la Búsqueda Dispersa Global (Figura 3).

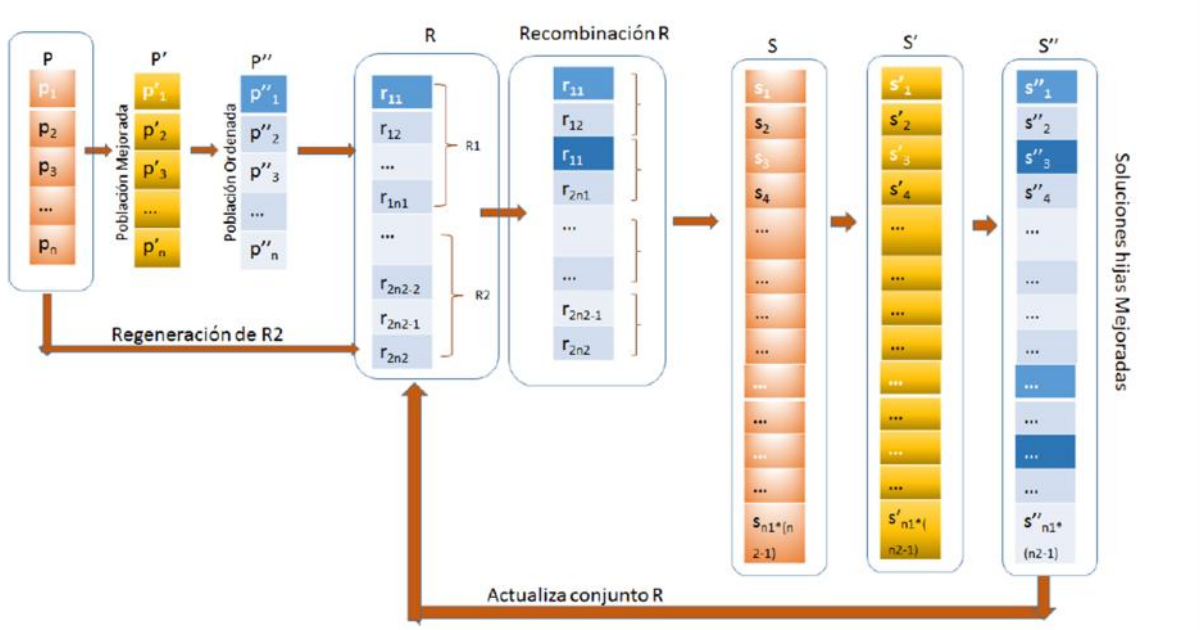

Figura 3: Búsqueda Dispersa Global.

4.2 BÚSQUEDA DISPERSA ESTRATIFICADA

En esta versión denominada Búsqueda Dispersa Estratificada (BDE) el mecanismo del algoritmo propuesto es similar a la versión anterior. La única diferencia es la forma de generar el subconjunto R_2 de las soluciones más diversas. Para ello, toma el conjunto de soluciones, lo divide en la cantidad de soluciones necesarias para formar el subconjunto R_2 es decir n_2 (n_2 = 8, en nuestro caso) y toma una solución representante de cada partición. En este caso elige como representante de cada partición la solución de peor calidad. El funcionamiento de esta versión es el siguiente:

Se genera la población inicial de tamaño *N(N = 100)* de soluciones de manera aleatoria. Estas soluciones se mejoran utilizando un algoritmo de búsqueda local. En este caso se aplica *Hill Climbing* (HC). Se ordena la población utilizando un método de ordenamiento (*QuickSort*). Se toman las n_1 *(n_1 = 8)* mejores soluciones para formar el subconjunto R_1 y las n_2 *(n_2 = 8)* más diversas soluciones para formar el subconjunto R_2 (en este caso utilizando un representante de cada partición como se explicó anteriormente). Los siguientes pasos se repiten mientras no se encuentre la condición de fin y se hayan generado nuevas soluciones: se recombinan todas las soluciones del conjunto *R*. A las soluciones hijas generadas de esa recombinación (método se recombinación Dpx) se las mejora utilizando un método de búsqueda local (HC), se ordenan las soluciones hijas con un método de ordenamiento (*QuickSort*). Luego se seleccionar las mejores soluciones y se remplazan las del subconjunto R_1 si las mismas son mejores en cuanto a su calidad. Se seleccionan las más diversas (en este caso un representante de cada partición) y se reemplazan a las del subconjunto R_2. Si no se ha alcanzado la condición de fin y no hay mejoras, es decir que el subconjunto R_1 no ha sido reemplazado por ninguna nueva solución, se regenera el subconjunto R_2, de la siguiente manera: se crea una nueva población de soluciones aleatorias de tamaño *N(N = 100)*, luego se ordenan esas soluciones y las n_2 más diversas (en este caso se toma una solución representante de cada partición) reemplazan a todas las soluciones del subconjunto R_2. El pseudocódigo es similar a la versión anterior ya que la diferencia está solamente en la generación y

actualización del subconjunto R_2. El pseudocódigo de la BDE es similar al de la BDG (Algoritmo 1) solo difiere en el método que utiliza en el paso 4, donde la generación del conjunto R_2 es estratificada.

En el siguiente esquema (Figura 4) se muestra el mecanismo para generar el subconjunto R_2. Para ejemplificar se muestra una Población ordenada (P) de 12 elementos. El tamaño de R es *8* por lo tanto el del subconjunto R_2 es *4*. Se divide el tamaño de la población con el tamaño del subconjunto R_2 y se toma de cada partición un representante, en este caso el de mayor valor de cada partición (suponiendo un problema de minimización).

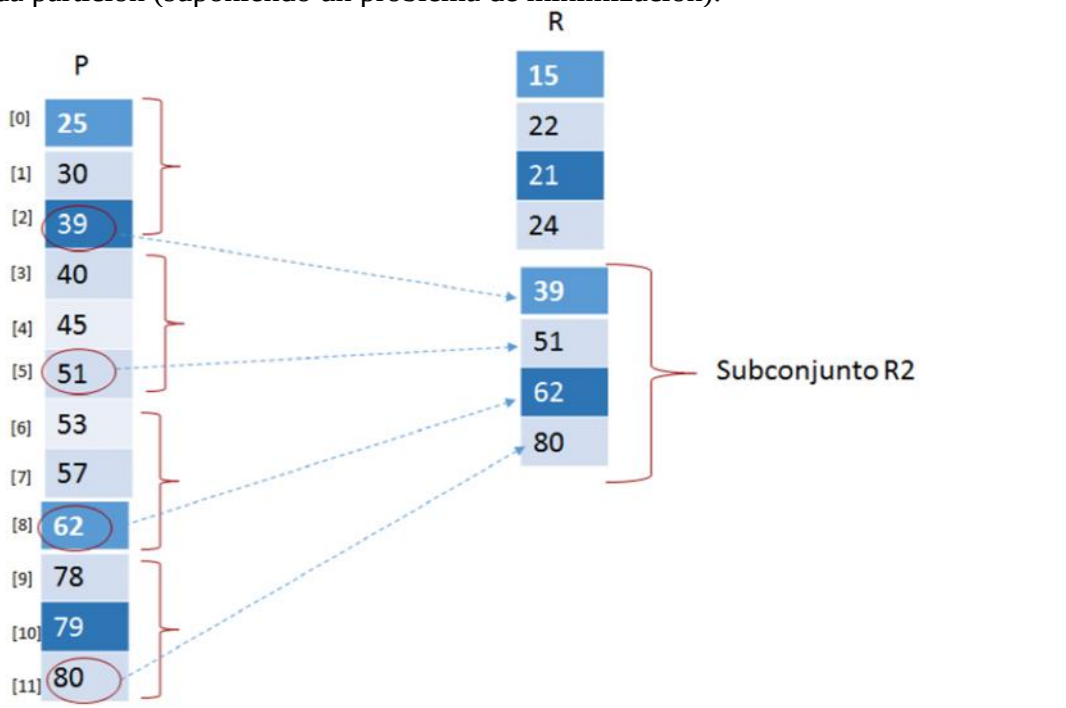

Figura 4: Búsqueda Dispersa Estratificada

Acabamos de presentar los dos algoritmos propuestos. En la siguiente sección describimos el subconjunto de problemas seleccionados para este trabajo.

5. PROBLEMAS DE OPTIMIZACIÓN COMBINATORIA

En esta sección describimos los distintos problemas de optimización combinatoria que hemos utilizado para el análisis comparativo, de los tiempos de ejecución, y número de evoluciones entre los dos algoritmos de búsqueda propuestos para la su resolución. Los mismos fueron elegidos ya que contienen características diferentes e interesantes para este estudio como ser epistasis, multimodalidad y problemas engañosos. A la hora de ser resueltos, permiten determinar si existen diferencias estadísticamente significativas de performance entre los dos algoritmos propuestos para la resolución de los mismos.

En la definición de algunos de ellos se utiliza la distancia de *Hamming*. La distancia de *Hamming* entre una cadena de *bits a* y otra *b* se mide con la ecuación:

$$d_{ab} = \sum_{i=1}^{l} a_i \otimes b_i$$

Por tanto, cadenas de bits idénticas tienen una distancia de *Hamming* de $d_{ab} = 0,0$, mientras que cadenas de *bits* completamente diferentes tendrán una distancia de $d_{ab} = l$, siendo l la longitud de las dos cadenas.

- **Problema COUNTSAT** (Droste et al. 2000). Es una instancia de MAXSAT. En COUNTSAT el valor de la solución se corresponde con el número de cláusulas (de entre todas las posibles cláusulas de *Horn* de tres variables) satisfechas por una entrada formada por *n* variables booleanas. Es fácil comprobar que el óptimo se consigue cuando todas las variables tienen el valor 1.

  En este estudio consideramos una instancia de $n = 20$ variables, y el valor de la solución óptima al problema es *6860*.

  La función COUNTSAT esta extraída de MAXSAT con la intención de ser muy difícil de resolver con GAs (Droste et al. 2000). Las entradas aleatorias generadas uniformemente tendrán $n = 2$ unos en término medio. Por tanto, los cambios locales decrementando el número de unos nos conducirán hacia entradas mejores, mientras que si se incrementa el número de unos se decrementará el valor de adecuación. En consecuencia, cabe esperar que un GA encuentre rápidamente la cadena formada por todos sus componentes a cero y tenga dificultades para encontrar la cadena compuesta por unos en todas sus posiciones.

- **Problema del Diseño de Códigos Correctores de Errores - ECC**. Consideremos una tupla *(n, M, d)* donde *n* es la longitud (número de *bits*) de cada palabra de código, *M* es el número de palabras de código, y *d* es la mínima distancia de *Hamming* entre dos palabras de código cualesquiera. El objetivo será obtener un código con el mayor valor posible de *d* (reflejando una mayor tolerancia al ruido y a los errores), y con valores de *n* y *M* fijados previamente.
  En este estudio consideramos una instancia donde *C* está formado por $M = 24$ palabras de longitud $n = 12$ *bits*. Lo que forma un espacio de búsqueda de tamaño $\left(\frac{4096}{24}\right)$, que es 10^{87} aproximadamente. La solución óptima para la instancia con $M = 24$ y $n = 12$ tendrá el valor de adecuación 0,0674 (Chen et al. 1998). En algunos de nuestros estudios, hemos implicado el espacio de búsqueda del problema reduciendo a la mitad *(M/2)* el número de palabras que forman el código, y la otra mitad se compone del complemento de las palabras encontradas por el algoritmo.

- **Problema de la Modulación en Frecuencia de Sonidos - FMS** (Tsutsui et al. 1997). Consiste en determinar los 6 parámetros reales $x = (a_1, w_1, a_2, w_2, a_3, w_3)$ del modelo de sonido en frecuencia, donde $\theta = 2®/100$. Este problema puede ser definido como un problema de optimización discreta o continua. En el caso que sea discreto, los parámetros están codificados en el rango [-6,4; + 6,35] con cadenas de 32 *bits*.

  El objetivo de este problema consiste en minimizar la suma del error cuadrático medio entre los datos de muestreo y los reales. Este problema es una función multimodal altamente compleja. Debido a la extrema dificultad para resolver este problema con alta precisión sin utilizar operadores específicos de optimización continua, el algoritmo se detiene cuando el error cae por debajo de 10^{-2}. La función de adecuación que tendremos que maximizar se corresponde a su máximo valor cuando EFMS = 0,0.

- **Problema del Máximo Corte de un Grafo - MAXCUT (**Khuri et al. 1994). Consiste en dividir un grafo ponderado $G = (V, E)$ en dos sub-grafos disjuntos $G_0 = (V_0, E_0)$ y $G_1 = (V_1, E_1)$ de manera que la suma de los pesos de los ejes que posean un extremo en V_0 y el otro en V_1 sea máxima. Para codificar una partición de los vértices utilizaremos una cadena binaria $(x, x_2, \ldots, x_n)$ donde cada dígito se corresponde con un vértice. De manera que si un dígito tiene valor 1, su vértice correspondiente estaría en el conjunto V_1, y si tuviera valor 0 el vértice respectivo pertenecería a V_0. Aunque es posible generar instancias de grafos diferentes de forma aleatoria, hemos utilizado tres instancias distintas del problema. Dos de ellas se corresponden con grafos generados aleatoriamente: uno disperso "cut20.01" y otro denso "cut20.09", ambos formados por 20 vértices. La otra instancia es un grafo escalable de 100 vértices. Las soluciones óptimas para estas instancias son 10,119812 para el caso de "cut20: 01", 56,740064 para "cut20.09" y 1077 para "cut100".

- **Problema Masivamente Multimodal - MMDP (**Goldberg et al. 1992). Fue específicamente diseñado para ser difícil de afrontar para los Algoritmos Evolutivos. Está compuesto de k sub-problemas engañosos (s_i) de 6 *bits* cada uno. El valor de cada uno de estos sub-problemas (*fitness* s_i) dependen del número de unos que tiene la cadena binaria que lo compone. Es fácil comprender por qué se considera a esta función engañosa, puesto que estas sub-funciones tienen dos máximos globales y un a tractor engañoso en el punto medio.

  En MMDP, cada sub-problema s_i contribuye al valor de *fitness* total en función del número de unos que tiene. El óptimo global del problema tiene el valor *k*, y se alcanza cuando todos los sub-problemas están compuestos de cero o seis unos. El número de óptimos locales es muy grande (22^k), mientras que sólo hay 2^k soluciones globales. Por tanto, el grado de multimodalidad viene dado por el parámetro *k*. Nosotros utilizaremos una instancia considerablemente grande de $k = 40$ sub-problemas. La función que trataremos de maximizar concierne con un valor máximo igual a 40.

- **Problema de las Tareas de Mínima Espera - MTTP**. Es un problema de planificación de tareas en el que cada tarea i del conjunto de tareas $T = \{1, 2, \ldots, n\}$ tiene una longitud l_i -el tiempo que tarda en ejecutarse- un límite de tiempo (o *deadline*) di -antes del cual debe ser planificada- y un peso w_i. El peso es una penalización que debe ser añadida a la función objetivo en el caso de que la tarea permanezca sin estar planificada. Las longitudes, pesos, y límites de tiempo de las tareas son todos números enteros positivos. Planificar la tarea de un sub-conjunto *S* de *T* consiste en encontrar el tiempo de comienzo de cada tarea en *S*, de forma que como mucho se realiza una tarea cada vez y que cada tarea termina de realizarse antes de su límite de tiempo. El objetivo del problema es minimizar la suma de los pesos de las tareas no planificadas.

  Hemos utilizado para nuestros estudios tres diferentes instancias **(**Khuri et al. 1994): "mttp20", "mttp100" y "mttp200", de tamaños 20, 100 y 200, y valores óptimos de 0,02439, 0,005 y 0,0025, respectivamente.

- **Problema OneMax** (Schaffer y Eshelman 1991). Es un problema muy simple que consiste en maximizar el número de unos que contiene una cadena de bits. Formalmente, este problema se puede describir como encontrar una cadena $x = \{x_1, x_2, \ldots, x_n\}$ con $x_i \in \{0, 1\}$, que maximice la función. La función f_{OneMax} tiene $(n + 1)$ posibles valores diferentes, que están distribuidos multimodalmente. Para las funciones aditivamente descomponibles (ADFS) la

distribución multimodal ocurre con bastante frecuencia **(**Goldberg et al. 1993). Para definir una instancia de este problema, sólo necesitamos definir la longitud de la cadena de bits (*n*). Para un n dado, la solución óptima al problema es una cadena de n unos, es decir, se les fija el valor uno a todos los bits de la cadena.

- **Problema P-PEAK** (De Jong et al. 1997). Es un generador de problemas multimodales. Un generador de problemas es una tarea fácilmente parametrizable, con un grado de dificultad que se puede afinar, de manera que nos permita generar instancias con una complejidad tan grande como queramos. Adicionalmente, el uso de un generador de problemas elimina la posibilidad de afinar a mano los algoritmos para un problema particular, permitiendo de esta forma una mayor justicia al comparar los algoritmos. Utilizando un generador de problemas, evaluamos nuestros algoritmos sobre un elevado número de instancias de problema aleatorias, ya que cada vez que se ejecuta el algoritmo resolvemos una instancia distinta. Por tanto, se incrementa el poder de predicción de los resultados para la clase de problemas como tal, no para instancias concretas.

  La idea de P-PEAKS consiste en generar *P* cadenas aleatorias de *N bits* que representan la localización de *P* cimas en el espacio de búsqueda. El valor de adecuación de una cadena se corresponde con la distancia de *Hamming* de esa cadena con respecto a la cima más próxima, dividida por *N*. Usando un mayor (o menor) número de cimas obtenemos problemas más (o menos) epistáticos. En nuestro caso hemos utilizado una instancia de *P = 100* cimas de longitud N = 100 bits cada una, representando un nivel de epistasis medio/alto (Alba et al. 2007). El máximo valor de *fitness* que podremos conseguir es 1,0.

6. DISEÑO DE EXPERIMENTOS Y RESULTADOS

En esta sección se presentan los experimentos llevados a cabo para analizar las dos versiones propuestas de la BD. El costo de la solución de un problema se analiza midiendo el número de evaluaciones de la función objetivo durante la búsqueda. La condición de parada para todos los algoritmos es encontrar una solución o lograr un máximo de evaluaciones de la función. A lo largo del documento los mejores resultados se muestran en negrita. La parametrización utilizada para ambos algoritmos es la siguiente: 16 individuos para el conjunto *P* (donde 8 individuos corresponden al subconjunto R_1 y *8* al subconjunto R_2). Se aplica el método de HC como búsqueda local, el operador DPX para la recombinación, *QuickSort* para el ordenamiento de las soluciones y el número máximo de evaluaciones es 1000000.

En todos los experimentos se han analizado las condiciones que deben cumplirse para utilizar pruebas paramétricas y pruebas no paramétricas para el análisis estadístico con una probabilidad de error *p = 0,05* con SPSS. Las diferencias estadísticamente significativas entre los algoritmos se muestran con los símbolos "(+)", mientras que las no significativas se muestran con "(-)". Todos los algoritmos se implementan en Java, y se ejecutan en un procesador Intel Core i5 de 2.53 GHz con Windows 7. Se realizaron 150 ejecuciones independientes para todos los algoritmos. Para realizar los test estadísticos tomamos las primeras 30 ejecuciones exitosas.

En primer lugar queremos corroborar que los problemas que elegimos son complejos y que también los algoritmos son adecuados para manejar los problemas. Para ello se implementó un algoritmo de Búsqueda Aleatoria (*Random Search*).

Problema	Búsqueda Aleatoria
COUNTSAT	57%
ECC	0%
FMS	0%
MMDP	0%
P-PEAK	0%
OneMax	0%
“cut20.01”	100%
“cut20.09”	100%
“cut100”	0%
“mttp20”	63%
“mttp100”	0%
“mttp200”	0%
Promedio	27%

Tabla 1: Porcentaje de éxito obtenido con un algoritmo de Búsqueda Aleatoria.

En la Tabla 1 se muestra el porcentaje de éxito en 150 ejecuciones independientes para el algoritmo de Búsqueda Aleatoria. Podemos observar un bajo promedio de tasa de éxito (27%). Sólo en cuatro problemas (COUNTSAT, “cut20.01”, “cut20.09”, y “mttp20”) el algoritmo de búsqueda aleatoria obtiene un valor de éxito diferente de 0. Estos resultados confirman que los problemas son lo suficientemente complejos y los algoritmos utilizados para resolverlos pueden gestionar el conjunto de problemas.

Problema	BDG	BDE
COUNTSAT	**95%**	14%
ECC	50%	**55%**
FMS	0%	**2%**
MMDP	0%	0%
P-PEAK	**100%**	**100%**
OneMax	**100%**	**100%**
“cut20.01”	**100%**	**100%**
“cut20.09”	**100%**	**100%**
“cut100”	0%	**3%**
“mttp20”	**100%**	**100%**
“mttp100”	62%	**100%**
“mttp200”	7%	**67%**
Promedio	60%	**62%**

Tabla 2: Porcentaje de éxito obtenido por los algoritmos BDG y BDE.

En el siguiente experimento se comparan los dos algoritmos propuestos BDG y BDE. En la Tabla 2 se muestra el porcentaje de éxito en 150 ejecuciones independientes. Podemos observar que la tasa de éxito para la segunda versión propuesta (BDE) es superior a la de la primer versión (BDG); 60% del éxito obtenido por BDG y el 62% por BDE). Se observa que con BDE se obtiene el 100% de éxito en 6 problemas (P-PEAK, OneMax, “cut20.01”, “cut20.09”, “mttp20”, “mttp100”).

R^+	R^-	*pvalue*
5	*1*	*0.345*

Tabla 3: *Wilcoxon rank test* para BDG y BDE en cuanto al porcentaje de éxito.

Como un análisis adicional se aplica una comparación por pares, prueba "*Wilcoxon signed ranks test*". El objetivo es comparar el desempeño de las dos versiones del algoritmo de Búsqueda Dispersa cuando se aplica a un conjunto común de problemas. En la Tabla 3 se muestran los valores R^+, R^- y el valor *p* de *Wilcoxon rank test*. R^+ representa la suma rangos para los problemas en los que el segundo algoritmo superó al primero, y R^- representa la suma de rangos para el contrario. Podemos observar que, si bien BDE supera a BDG en la mayoría de los problemas, las diferencias no son estadísticamente significativas.

Problema	BDG		BDE	
	N°Ev.	Tiempo*[ms]*	N°Ev.	Tiempo *[ms]*
ECC	159600	1838.5	**153640**	**1785.5**
P-PEAK	**90550**	11604.5	93365	**11278**
OneMax	453630	**1756**	**410065**	2077
"cut20.01"	21465	31.5	**13305**	**20.5**
"cut20.09"	41485	77	**26485**	**40**
"mttp20"	14930	**10**	**11995**	**10**
"mttp100"	203770	264	**171650**	**256.5**

Tabla 4: Resultados obtenidos por BDG y BDE en cuanto al número de evaluaciones y tiempo requerido.

En la Tabla 4 se muestra para cada una de las versiones propuestas, el número de evaluaciones y el tiempo (en milisegundos requerido para encontrar el valor óptimo.). Podemos observar que para cinco de los siete problemas (ECC, OneMax, "cut20.01", "cut20.09"y "mttp100") el menor número de evaluaciones es obtenido por BDE. En cuanto al tiempo, los resultados que obtuvimos son prácticamente similares, solo que en este caso en seis de los siete problemas (ECC, P-PEAK, "cut20.01", "cut20.09", "mttp20"y "mttp100") los menores tiempos de ejecución se obtuvieron por la BDE.

En la Tabla 5 se muestran los resultados de la prueba *Kolmogorov-Smirnov* aplicada a los resultados obtenidos de número de evaluaciones y el tiempo en milisegundos. Donde el símbolo "*" indica que los resultados obtenidos por el algoritmo no cumple con la condición de normalidad.

Problema	BDG		BDE	
	N°Ev.	Tiempo *[ms]*	N°Ev.	Tiempo*[ms]*
ECC	(*)0.000	(*)0.025	(*)0.000	(*)0.000
P-PEAK	(*)0.000	0.176	0.200	0.200
OneMax	0.300	(*)0.002	0.186	0.087
"cut20.01"	(*)0.018	0.165	(*)0.000	(*)0.002
"cut20.09"	(*)0.000	(*)0.002	(*)0.000	(*)0.000
"mttp20"	(*)0.000	(*)0.002	(*)0.000	0.066
"mttp100"	(*)0.000	(*)0.000	(*)0.007	0.108

Tabla 5: Prueba de *Kolmogorov-Smirnov* aplicado a variables de "N° de Evaluaciones" y "Tiempo" de BDG y BDE.

En la Tabla 6 se muestran los resultados obtenidos al realizar el *test* de *Levene*, para determinar si el conjunto de datos a analizar posee o no homocedasticidad, donde el símbolo "*" indica que el resultado obtenido para el conjunto de datos no cumple con la condición de homocedasticidad. Recordemos que para aplicar test paramétricos los resultados deben cumplir las siguientes tres condiciones: independencia, normalidad y homocedasticidad. Al tratarse de ejecuciones independientes de los algoritmos la primera condición está satisfecha. Deben cumplirse las dos condiciones restantes para la aplicación de test paramétricos.

Problema	Estadístico (p-valor)	
	N°Ev.	Tiempo*[ms]*
ECC	0.393	0.361
P-PEAK	(*)0.000	(*)0.012
OneMax	0.565	0.066
"cut20.01"	0.689	0.679
"cut20.09"	0.812	0.580
"mttp20"	(*)0.021	0.726
"mttp100"	(*)0.000	(*)0.000

Tabla 6: Test de *Levene* aplicado a variables de "N° de Evaluaciones" y "Tiempo" de BDG y BDE.

Los resultados obtenidos de los estadísticos de *Levene* y *Kolmogorov-Smirnov*, determinamos que para seis de los siete resultados se deben aplicar test no paramétricos (*Mann Whitney test*) y para los resultados obtenidos por los algoritmos en OneMax se debe aplicar un test paramétrico (*t-test*). Se utiliza el signo (+) para especificar que existen diferencias estadísticamente significativas entre los resultados obtenidos en los problemas y (-) en el caso contrario.

Si se observan los resultados de la Tabla 7 en cuanto al número de evaluaciones, que 3 de los 7 problemas analizados en esta oportunidad presentan diferencias estadísticamente significativas los resultados obtenidos por BDE con respecto a BDG. Tanto en la instancia OneMax como en "mttp20"y "mttp100", los menores valores fueron obtenidos con el algoritmo BDE. Si se tiene en cuenta la variable correspondiente al tiempo podemos observar que en seis de los siete problemas BDE fue la que logró el menor de los valores. No obstante las diferencias no son estadísticamente significativas. Si existe diferencia estadísticamente significativa de BDG con respecto a BDE para la instancia OneMax. A continuación se muestran gráficamente los casos más significativos.

Problema	BDG		BDE		Mann-Whitney/ T- test	
	N°Ev.	Tiempo *[ms]*	N°Ev.	Tiempo*[ms]*	N°Ev.	Tiempo*[ms]*
ECC	159600	1838.5	**153640**	**1785.5**	(-)	(-)
P-PEAK	**90550**	11604.5	93365	**11278**	(-)	(-)
OneMax	453630	**1756**	**410065**	2077	(+)	(+)
"cut20.01"	21465	31.5	**13305**	**20.5**	(-)	(-)
"cut20.09"	41485	77	**26485**	**40**	(-)	(-)
"mttp20"	14930	**10**	**11995**	**10**	(+)	(-)
"mttp100"	203770	264	**171650**	**256.5**	(+)	(-)

Tabla 7: *Test de Mann-Whitney y T- test* según corresponda, aplicado a variables de "N° de Evaluaciones" y "Tiempo" de BDG y BDE.

En las Figuras 5 y 6 se muestra la gráfica *boxplot* de los resultados de las variables "N° de Evaluaciones" y "Tiempo" respectivamente de BDG y BDE aplicados al problema OneMax. Se observa que para las dos variables analizadas los mejores resultados (los menores) son obtenidos, en el primer caso (número de evaluaciones) por BDG y en el segundo caso (tiempo) por BDE.

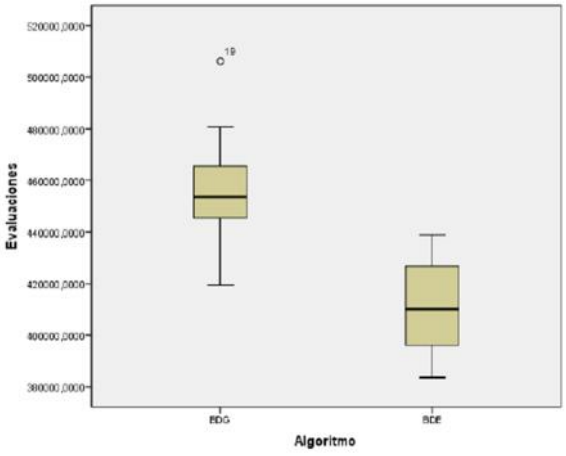

Figura 5: *Boxplot* de los resultados de la variable "N° de Evaluaciones" de BDG y BDE para resolver el problema OneMax.

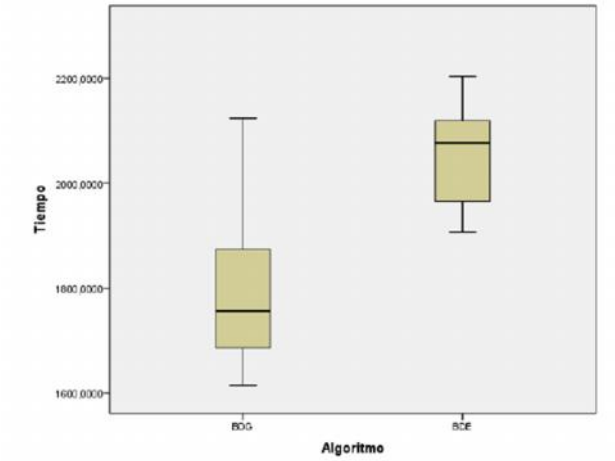

Figura 6: *Boxplot* de los resultados de la variable de "Tiempo" de BDG y BDE para resolver el problema OneMax.

Los siguientes resultados corresponden a las instancias "mttp20" y "mttp100" respectivamente, tanto en la primera de ellas (Figura 7) como en la segunda (Figura 8) se analiza la variable "N° de Evaluaciones". Haciendo referencia a la Figura 7 podemos observar que los mejores resultados son obtenidos con BDE y además sabemos que las diferencias son estadísticamente significativas en relación a su par, mientras que en la Figura 8 claramente la segunda versión del algoritmo es la que obtuvo mejores resultados (más robusto y compacto alrededor de la mediana), esto se aprecia a simple vista, lo que nos lleva a decir que para los problemas "mttp20" y "mttp100" el algoritmo de mayor rendimiento es BDE.

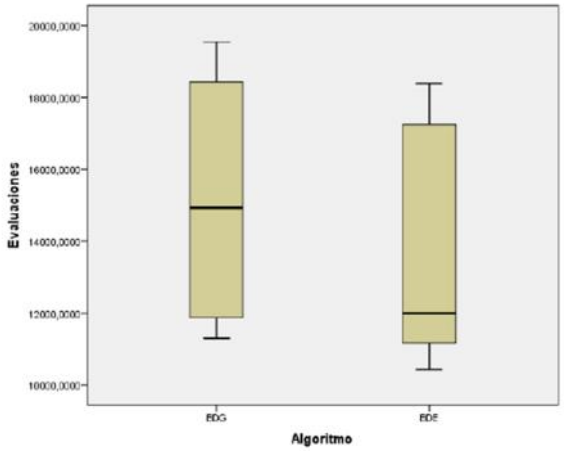

Figura 7: *Boxplot* de los resultados de la variable "N° de Evaluaciones" de dos versiones de un algoritmo de Búsqueda Dispersa para resolver el problema "mttp20".

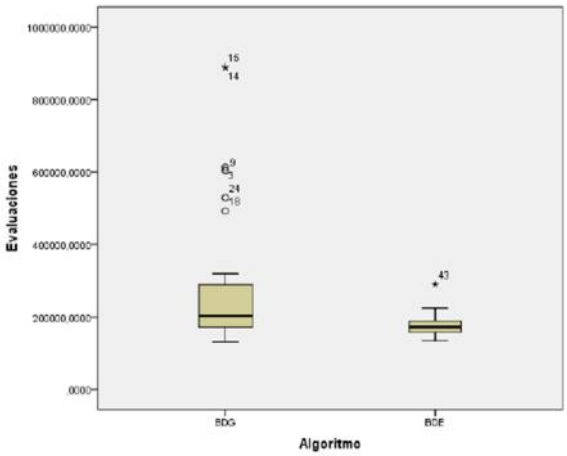

Figura 8: *Boxplot* de los resultados de la variable "N° de Evaluaciones" de dos versiones de un algoritmo de Búsqueda Dispersa para resolver el problema "mttp100".

A. CONCLUSIONES

En el presente trabajo de investigación se ha llevado a cabo el estudio del comportamiento de dos variantes de un algoritmo de Búsqueda Dispersa (BDG y BDE). Para ello se han seleccionado una serie de problemas de diversas características (COUNTSAT, ECC, FMS, “cut20.01”, “cut20.09”, “cut100”, MMDP, “mttp20”, “mttp100”, “mttp200”, OneMax y P-PEAK). De los resultados obtenidos se realizó un estudio estadístico detallado sobre las variables de performance, número de evaluaciones y tiempo de ejecución. Luego de realizar todos los análisis correspondientes concluimos que para los problemas estudiados los mejores resultados han sido obtenidos en su gran mayoría con la segunda versión del algoritmo de Búsqueda Dispersa propuesta en este trabajo, denominada BDE. Esto se ve reflejado en los resultados estadísticos obtenidos, en cuatro casos se observaron diferencias estadísticamente significativas con respecto a BDG. Además, en los casos donde no se encontraron diferencias estadísticamente significativas BDE fue la versión que obtuvo los mejores valores.
Por lo tanto, BDE será la versión a utilizar en trabajos futuros. Además se pretende aplicar esta versión en estudios comparativos con otras metaheurísticas poblacionales, como por ejemplo, con respecto a un algoritmo genético celular (cGA), y utilizar BDE para hibridar otras metaheurísticas.

7. AGRADECIMIENTOS

Se agradece la cooperación del equipo de proyecto del LabTEm y la Universidad Nacional de la Patagonia Austral, de los cuales se recibe apoyo continuo.

REFERENCIAS

Alba E., Dorronsoro B., Luna F., Nebro A. J., Bouvry P., y Hogie L. 2007. A cellular multi-objective genetic algorithm for optimal broadcasting strategy in metropolitan MANETs. Computer Communications, 30(4), 685-697.

Alba E. y Luque G. 2003. Algoritmos híbridos y paralelos para la resolución de problemas combinatorios. Actas del Segundo Congreso Español de Metaheurísticas y Algoritmos Evolutivos y Bioinspirados (MAEB’03), Gijón.

Chen H., Flann N. S., y Watson D. W. 1998. Parallel genetic simulated annealing: a massively parallel simd algorithm. Parallel and Distributed Systems, IEEE Transactions on, 9(2):126-136.

De Jong K. A., Potter M. A., y Spears W. M. 1997. Using problem generators to explore the effects of epistasis. En ICGA, pp 338–345. Citeseer.

Droste S., Jansen T., y Wegener I. 2000. A natural and simple function which is hard for all evolutionary algorithms, volume 4. IEEE.

Gendreau M. y Potvin J. Y. 2010. Handbook of Metaheuristics, Springer US.

Glover F. 1977. Heuristics for integer programming using surrogate constraints. Decision Sciences, 8(1):156–166.

Glover F. 1986. Future paths for integer programming and links to artificial intelligence. Computers Operations Research.

Glover F. 1989. Tabu search-part i. ORSA Journal on computing, 1(3):190–206.

Glover F. 1994. Tabu search for nonlinear and parametric optimization (with links to geneticalgorithms). Discrete Applied Mathematics, 49(1):231–255.

Glover F. 1998. A template for scatter search and path relinking. In Artificial evolution, pp 13–54. Springer-Verlag.

Glover F., Laguna M., y Martí R. 2000. Fundamentals of scatter search and path relinking. Control and cybernetics, 29:653–684.

Glover F., Laguna M., y Martí R. 2003. Scatter search. In Advances in evolutionary computing. Springer.

Goldberg D. E., Deb K., Hillol K., y Harik G. 1993. Rapid, accurate optimization of difficult problems using messy genetic algorithms. In Proceedings of the Fifth International Conference on Genetic Algorithms, (Urbana, USA), pp 59–64.

Goldberg D. E., Deb K., Hillol K., y Horn J. 1992. Massive multimodality, deception, and genetic algorithms. pp 37–46.

Holland J. H. 1992. Algoritmos genéticos. Investigación y Ciencia, 192:38–45.

Khuri S., Bäck T., y Heitkötter J. 1994. An evolutionary approach to combinatorial optimization problems. In ACM Conference on Computer Science, pp 66–73. Citeseer.

Laguna, M., Gortázar, F., Gallego, M., Duarte, A., y Martí, R. 2014. A black-box scatter search for optimization problems with integer variables. Journal of Global Optimization, 58(3):497–516.

Li, K. and Tian, H. 2015. A de-based scatter search for global optimization problems. Discrete Dynamics in Nature and Society.

Martí R. 2003. Procedimientos metaheurısticos en optimización combinatoria. Matemátiques, Universidad de Valencia, 1(1):3–62.

Martí R., y Laguna M. 2003. Scatter Search: Diseño Básico y Estrategias avanzadas. Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial, 7(19), 123-130.

Osman H. I. y Kelly J. P. 1996. Meta-heuristics: an overview. In Meta-Heuristics, pp 1–21. Springer.

Reeves C. R. 1993. Modern heuristic techniques for combinatorial problems. John Wiley & Sons, Inc.

Sait S. M. y Youssef H. 1999. Iterative computer algorithms with applications in engineering: solving combinatorial optimization problems. IEEE Computer Society Press.

Schaffer D y Eshelman L. J. 1991. On crossover as an evolutionarily viable strategy. In ICGA, volume 91, pp 61–68.

Tsutsui S., Ghosh A., Corne D., y Fujimoto Y. 1997. A real coded genetic algorithm with an explorer and an exploiter populations. In ICGA, pp 238– 245.

Van Laarhoven P. JM. y Aarts E. HL. 1987. Simulated annealing. Springer.