Algoritmo memético para el Problema del Particionado de Grafos en ejecuciones a largo plazo

Emmanuel Romero Ruiz[1], Carlos Segura[2]
[1]Departamento de Matemáticas, Universidad de Guanajuato
[2]Área de Computación, Centro de Investigación en Matemáticas
Correos electrónicos: emmanuel.romero@cimat.mx
carlos.segura@cimat.mx (Autor de Contacto)

Resumen

El problema del Particionado de Grafos (GPP — Graph Partitioning Problem) es un problema NP-Difícil cuyo objetivo es particionar los nodos de un grafo en k conjuntos de forma que se minimice el número de aristas que unen vértices localizados en diferentes conjuntos, a la vez que se cumplen restricciones relativas al tamaño de los conjuntos. En los últimos años, el desarrollo de metaheurísticas poblacionales que incorporan un control explícito de la diversidad ha permitido realizar avances significativos al tratar diversos problemas de optimización combinatoria como el problema de coloreado de grafos o el de asignación de frecuencias. A pesar de que el GPP es un problema clásico, el desarrollo de metaheurísticas poblacionales con control de diversidad para el GPP es un tema prácticamente inexplorado. Por ello, a partir de la hipótesis de que un algoritmo memético con gestión de diversidad ayudaría a mejorar el estado del arte para el GPP, se desarrollaron nuevos algoritmos que efectivamente han permitido mejorar las soluciones encontradas hasta la fecha para una gran cantidad de casos. El trabajo presentado es un resumen de la tesis que ganó el Premio Mixbaal a la mejor tesis de licenciatura en matemáticas aplicadas, en la edición del año 2019. Este premio es otorgado por la Sociedad Mexicana de Computación Científica y Sus Aplicaciones.

Palabras clave: Particionado de Grafos; Algoritmos Meméticos; Control de Diversidad; Búsqueda Tabú.

Introducción

El Problema del Particionado de Grafos (GPP — Graph Partitioning Problem) es un problema NP-Difícil cuyo objetivo es particionar el conjunto de nodos de un grafo en k conjuntos de forma que se minimice el número de aristas que unen vértices localizados en diferentes conjuntos, a la vez que se cumplen restricciones relativas al tamaño de los conjuntos. Este problema, además de ser importante en el ámbito académico donde es común su utilización para realizar comparativas entre diferentes optimizadores, tiene también aplicaciones prácticas. Así, se ha utilizado para determinar la manera en que se agrupan componentes en un conjunto de chips cuando se busca minimizar la comunicación entre elementos de diferentes chips, así como en la paralelización de estrategias de elemento finito, donde se utiliza para distribuir las variables del problema entre los diferentes procesadores con el objetivo de minimizar la información compartida entre dichos procesadores. Bajo la hipótesis P $\neq$ NP, no es posible desarrollar resolutores exactos para el GPP que tengan complejidad polinomial con respecto al número de nodos del grafo. Por ello, y dado que cada vez se quiere lidiar con grafos de mayor tamaño, hasta la fecha un gran número de investigadores han trabajado el problema de manera aproximada, desarrollándose estrategias de múltiples tipos que abarcan desde algoritmos de aproximación, hasta heurísticas de trayectoria y poblacionales.

Las metaheurísticas poblacionales han sido una de las estrategias más exitosas, siendo los métodos que han permitido alcanzar los mejores resultados conocidos hasta la fecha en gran parte de grafos de gran tamaño que se usan para validar los algoritmos propuestos en el área de GPP. En lo referente a aplicación de algoritmos poblacionales en el ámbito del GPP, a pesar de toda la literatura que hay al respecto, no se han aplicado aún algunos de los últimos avances. Por ejemplo, no hay trabajos que apliquen algoritmos poblacionales con control explícito de diversidad al GPP. Teniendo esto en cuenta, en la tesis en la que este trabajo se basa se analizan los algoritmos poblacionales que han sido aplicados con mayor éxito al GPP y se proponen varias extensiones a los mismos incluyendo la utilización de un control explícito de diversidad. El propósito principal es desarrollar nuevos algoritmos poblacionales capaces de obtener soluciones de alta calidad a largo plazo, es decir, que sean capaces de obtener mejores resultados que cualquiera de las propuestas existentes aunque ello implique que a corto plazo no obtengan soluciones de alta calidad.

Definición matemática del GPP

En esta sección se incluye una definición formal de la versión del GPP [2] tratada en este trabajo:

* Sea $G = (V, E, w)$ un grafo no dirigido donde V es el conjunto de nodos, E es el conjunto de aristas y $w : E \to \mathbb{R}_{>0}$ es una función de pesos.
* En lo sucesivo $n = \|V\|$ y $m = \|E\|$.
* Sea k un entero positivo fijo que representa el número de clases en la partición deseada, y $\epsilon > 0$ un número real relacionado con la restricción de balance del problema.
* El espacio de soluciones admisibles para el problema es el conjunto de las particiones $\Delta = \{S = \{S_1, S_2, ..., S_k\}$ tal que S es una partición de V en k conjuntos y $\|S_i\| \leq L_{max} := (1+\epsilon)\lceil n/k \rceil$ para todo $i \in \{1, 2, .., k\}$ $\}$.
* Finalmente, la función a minimizar en el GPP es $cost(S) = \sum_{i<j} w(E_{ij})$ donde $E_{ij} = \{(u, v) \in E : u \in S_i, v \in S_j\}$ y $w(C) = \sum_{e \in C} w(e)$.

Nótese que existen otras variantes más generales, que por ejemplo establecen pesos en los nodos u otras restricciones relativas al balanceo [11]. Sin embargo, los desarrollos realizados en la tesis fueron exclusivos para la variante anterior. La validación experimental se llevó a cabo con las instancias presentes en el TGPA (The Graph Partitioning Archive) de Chris Walshaw [1]. Por ello, se debe considerar que el problema tratado está más limitado que el dado por la definición anterior ya que:

- En las instancias del TGPA, todos los pesos son iguales a 1, por lo que la función objetivo es el número de aristas en conflicto.
- Se trabajó exclusivamente con el valor de $\epsilon = 0$.

Si se quisiera tratar casos para valores diferentes de ϵ o grafos que contengan pesos diferentes de 1 sería necesario realizar algunas modificaciones en el algoritmo propuesto. Sin embargo, esto queda fuera del alcance de este trabajo.

Trabajo relacionado

Esta sección está dedicada a la revisión de trabajos que están altamente relacionados con nuestro enfoque. Primero, entre el gran conjunto de Algoritmos Evolutivos (AEs) que se han usado para la resolución del GPP, se discuten aquellos que han obtenido los resultados más prometedores, a la vez que aquellos que comparten varias características con nuestra propuesta. Además, puesto que nuestra propuesta está basada en el control de la diversidad, se incluye un resumen de las técnicas más populares que se han propuesto para mitigar los efectos de la convergencia prematura. Algunas de esas técnicas se usan para validar nuestra propuesta.

Algoritmos evolutivos para el GPP

Para lidiar con el GPP [3] se han diseñado un gran número de metaheurísticas poblacionales. Entre ellas, los AEs son los enfoques más populares. Desde las propuestas iniciales [4], fue claro que incorporar un procedimiento para intensificar en las regiones localizadas por el AE era bastante importante. En la mayoría de los algoritmos iniciales, se toma en cuenta la heurística propuesta por Kernighan y Li. Esta heurística es un proceso barato computacionalmente que obtiene particiones competitivas. Con el incremento del poder computacional, se han propuesto formas más complejas de intensificar. En algunos casos, se integran AEs con estrategias basadas en búsquedas de trayectoria [8], mientras que en otros casos, se usan búsquedas locales por escalada con grandes vecindades [21]. Estas clases de enfoques implican el uso de recursos adicionales, pero actualmente su uso es imprescindible para alcanzar las mejores soluciones conocidas al considerar grafos de gran tamaño.

Se han dedicado también muchos esfuerzos al diseño de operadores de cruce. En los enfoques iniciales, se aplicaron operadores simples, como el uniforme y/o el cruce de n-puntos [17]. Sin embargo, rápidamente fue claro que estos operadores eran bastante destructivos. Particularmente, tenían la tendencia de crear particiones muy desbalanceadas, implicando que después de los mecanismos de balance, la descendencia pudiera compartir sólo una pequeña cantidad de las características de los padres [3]. En [5] se propone un intento de evitar este problema. En este caso, los vértices que comparten la misma clase en ambos padres mantienen su clase en los descendientes. Luego, los vértices restantes se fijan usando una heurística constructiva voraz que asegura el balance apropiado. Un elemento que no se toma en cuenta en los operadores previos es que el identificador específico de la clase de un vértice no es lo importante. Las características importantes están relacionadas con el conjunto de vértices que compartan la misma clase. Tomando esto en cuenta, se proponen operadores de cruce más complejos [8] basados en heredar estas clases de características. El principio tras estos esquemas es transmitir grandes conjuntos de vértices que compartan la misma clase de padres a hijos. En [8] se usa un enfoque voraz basado en maximizar intersecciones y previamente en [7] se discuten propuestas no voraces. De cualquier manera, estas ideas se aplicaron al problema del coloreado de grafos y no al GPP. Puesto que esta clase de operadores han reportado resultados prometedores, nuestra propuesta toma en cuenta este principio. Particularmente, algunas de las ideas sugeridas en [7] inspiraron el desarrollo de nuestro nuevo operador de cruce. Sin embargo, nuestro operador además induce la creación de particiones relativamente balanceadas en la descendencia, lo que es una características importante en el GPP y no necesariamente en el problema del coloreado de grafos.

Finalmente un problema importante que no ha sido estudiado en profundidad es el manejo de la diversidad. Sin embargo, algunos análisis parecen indicar que el manejo adecuado de la diversidad puede ser importante para el funcionamiento adecuado de los optimizadores en el área del GPP. Por ejemplo, en [9], el número de generaciones se limitó a 200 y los autores argumentan que después de este número de generaciones, todos los miembros de la población se encontraban en una misma región y que mejoras con ejecuciones más largas eran poco probables. Adicionalmente, en muchos casos, se incluyen algunas acciones para retrasar la convergencia. Por ejemplo, en [4] el descendiente sustituye al padre más similar, mientras que recurrir a mecanismos de reinicio cuando se detecta convergencia prematura es otra de las estrategias más utilizadas [19].

Control de diversidad en algoritmos evolutivos

Dado que en nuestro enfoque se incorpora una forma explícita de controlar la diversidad, esta subsección discute algunas de las estrategias más populares que han sido diseñadas con la intención de evitar la convergencia prematura. Particularmente, se describen las técnicas que han sido usadas en la validación experimental de nuestra propuesta. Se remite a los lectores a [26] para consultar descripciones más detalladas de estas y otras técnicas relacionadas. Nótese que se han propuesto un gran número de técnicas para tratar con convergencia prematura [20]. Los métodos se clasifican usualmente con base en la componente del AE que modifican en los siguientes grupos [26]: *basados en selección, basados en población, basados en mutación o cruce, basados en función objetivo y basados en reemplazamiento*. Los métodos basados en reemplazamiento han reportado resultados prometedores en varios problemas de optimización [25] [23]. Dado que nuestra propuesta pertenece a esta categoría, se consideran otros tres esquemas basados en reemplazamiento para este artículo:

- *Restricted Tournament Selection* (RTS) [10] es un esquema popular en el cual después de que un nuevo individuo (O) es creado, CF individuos de la población son seleccionados aleatoriamente. Luego, el mejor individuo entre (O) y el más similar a éste entre los del conjunto seleccionado sobrevive. Nótese que este esquema pertenece al grupo de estrategias de amontonamiento [18].
- En el Algoritmo Híbrido Genético con Control de Diversidad Adaptativo [27] (HGSADC - Hybrid Genetic Search with Adaptative Divesity Control), los individuos son ordenados con base en su contribución a la diversidad y a su costo original. Luego, dichas ordenaciones de los individuos son usadas para calcular una puntuación usando dos parámetros (N_{Close} y N_{Elite}) la cual es usada para determinar a los individuos sobrevivientes.
- El Algoritmo "Contribution Diversity - Replace Worst" (CD/RW) [16] es un algoritmo de estado estacionario en el que los nuevos individuos tratan de reemplazar a otros que sean peores que ellos tanto al considerar contribución a la diversidad como cali-

Algoritmo 1 Algoritmo Memético tipo Lamarck

1: Generar una población inicial.
2: **while** No se satisfagan las condiciones de paro **do**
3: *Evaluar* todos los individuos en la población.
4: *Selecciona Padres* de entre la población para el conjunto de reproducción Ω a través de un operador $S(.)$.
5: **for** Individuos en Ω **do**
6: *Evolucionar* uno o varios descendientes a través de operador $C(.)$.
7: *Mutar* descendientes con operador $M(.)$
8: *Realizar* Aprendizaje Individual en descendientes a través de un operador $IL(.)$ conforme al espíritu del aprendizaje Lamarckiano, es decir, actualizando los valores de las variables.
9: **end for**
10: *Reemplazar* la población actual a partir de los descendientes y la población anterior usando $R(.)$.
11: **end while**

dad. En caso de que eso no sea posible se utiliza la técnica RW, consistente en reemplazar al peor individuo.

Nótese que, además de las técnicas con control de diversidad, incorporamos un esquema generacional con elitismo (GEN_ELIT) [6] para nuestra validación experimental. Este esquema no incorpora un mecanismo especial para retrasar la convergencia. Sin embargo, fue tomado en consideración porque es una estrategia muy aplicada en el campo de los AEs. En el GEN_ELIT la nueva población contiene la descendencia y la mejor solución de la generación anterior.

Propuesta algorítmica: memético para el GPP

Este sección está dedicada a describir nuestra propuesta para el GPP. Se presenta a detalle la nueva propuesta algorítmica, cada uno de los componentes usados y la manera de tener implementaciones eficientes de ellos. Nuestra propuesta es un AE que incorpora una búsqueda de trayectoria. El algoritmo se deriva de la estructura de un Algoritmo Memético estándar [13] (véase Algoritmo 1). Sin embargo, para satisfacer las restricciones del GPP es necesario modificar el esquema general incorporando una estrategia de balanceo, ya que tras realizar el proceso de creación de hijos, las nuevas particiones pueden estar desbalanceadas, y por tanto, no ser válidas. Con el objetivo de caracterizar completamente nuestra propuesta, se deben definir los operadores de cruce $C(.)$, mutación $M(.)$, el operador de balanceo, el aprendizaje individual $IL(.)$, la estrategia de selección de padres $S(.)$ y el operador de reemplazamiento $R(.)$. Las siguientes subsecciones contienen detalles de cada una de estas componentes.

Operador de cruce

Como es habitual, el operador de cruce toma dos individuos y genera dos nuevos con características similares a los padres. Muchos de los operadores de cruce que han sido usados para el GPP están inspirados en aquellos desarrollados para el problema del coloreado de grafos. Para nuestra propuesta se ha diseñado un nuevo operador de cruce, el Operador de Cruce Basado en el Algoritmo Húngaro (HBX - Hungarian Based Crossover) — ver Algoritmo 2 — que está inspirado en uno propuesto en [7] para el problema del coloreado de grafos. Particularmente, HBX se basa en el principio de maximizar el número de nodos que comparten la misma clase en padres y descendientes.

El HBX funciona como sigue. Sean $I_1 = \{S_1^1, S_2^1, ..., S_k^1\}$ y $I_2 = \{S_1^2, S_2^2, ..., S_k^2\}$ los padres seleccionados como entrada para el operador de cruce. El método Húngaro es un algoritmo de optimización popular que resuelve, en tiempo polinomial, el problema de asignación en grafos bipartitos pesados, es decir, calcula el máximo o mínimo emparejamiento pesado. El algoritmo comienza construyendo un grafo completo bipartito (G) con $2 \times k$ vértices, de forma que los primeros k vértices están asociados a los subconjuntos de I_1, mientras que los últimos corresponden a los subconjuntos de I_2. Las aristas conectan cualquiera de los primeros k vértices con cualquiera de los últimos

Algoritmo 2 Operador de reproducción basado en el algoritmo Húngaro

1: Sea elg_r un vector booleano para las filas "elegibles".
2: Sea elg_c un vector booleano para las columnas "elegibles".
3: Sea blc_r un vector booleano para las filas "bloqueadas".
4: Sea blc_c un vector booleano para las columnas "bloqueadas".
5: Fijar los vectores elg a verdadero.
6: Fijar los vectores blc a falso.
7: **for** i en $\{1, 2, ..., n\}$ **do**
8: **if** $i \equiv 0, mod(2)$ **then**
9: $L = -1$; $B_L = \emptyset$.
10: **for** l tal que $elg\_c[l]$ es verdadero (de forma aleatoria) **do**
11: $B_l = \{P_{i,l}$ no eliminadas$\} \cup \{P_{\sigma^{-1}(l),m}$ tal que $blc\_c[m]$ es verdadero $\}$.
12: Si ($|B_l| > |B_L|$) fijar $L = l$.
13: **end for**
14: $T_j = B_L$.
15: Elimina todos los elementos de T_j de la matriz P.
16: $elg\_c[L] =$ falso.
17: $elg\_r[\sigma^{-1}(L)] =$ falso.
18: $blc\_r[\sigma^{-1}(L)] =$ verdadero.
19: **else**
20: $L = -1$; $B_L = \emptyset$.
21: **for** l tal que $elg\_r[l]$ es verdadero (en forma aleatoria) **do**
22: $B_l = \{P_{l,i}$ no eliminados$\} \cup \{P_{m,\sigma(l)}$ tal que $blc\_r[m]$ es verdadero $\}$.
23: If ($|B_l| > |B_L|$) fijar $L = l$.
24: **end for**
25: $T_j = B_L$.
26: Eliminar de P todos los elementos de T_j.
27: $elg\_r[L] =$ falso.
28: $elg\_c[\sigma(L)] =$ falso.
29: $blc\_c[\sigma(L)] =$ verdadero.
30: **end if**
31: **end for**

k, quedando el peso de cada arista establecido como el tamaño de la intersección de los conjuntos asociados con cada vértice. La permutación σ obtenida con la aplicación del Algoritmo Húngaro a G maximiza $\sum_{i=1}^{k} \|S_i^1 \cap S_{\sigma(i)}^2\|$.

Principales propiedades del HBX

Previo a dar el procedimiento específico, es importante discutir algunas de las propiedades que tendrán los descendientes, que se consideraron como propiedades deseables al momento de diseñar el HBX. Sea P la matriz de intersección definida como $P_{i,j} = S_i^1 \cap S_j^2$ y sea $T = \{T_1, T_2, ..., T_k\}$ el hijo creado, entonces T tiene las siguientes propiedades:

1. Para cada T_j, existe un único i_j tal que $S_{i_j}^1 \cap S_{\sigma(i_j)}^2 \subset T_j \subset S_{i_j}^1 \cup S_{\sigma(i_j)}^2$.
2. $P_{i,j} \subset T_l$ para todos i, j y algún l.
3. Todo T_l es la unión de exactamente k $P_{i,j}$ distintos.

La razón tras el uso de σ es que el Algoritmo Húngaro da el máximo emparejamiento entre los conjuntos de ambas particiones, lo cual está relacionado con el principio enunciado de maximizar el número de nodos que comparten la misma clase en padres y descendientes. Esto significa que S_i^1 es de alguna manera similar a $S_{\sigma(i)}^2$ y su intersección es susceptible a tener muchos elementos. La propiedad 1) establece que estas intersecciones se usan como el núcleo de los conjuntos en la nueva partición. También establece que los elementos son seleccionados exclusivamente de uniones de los subconjuntos considerados. Tomando estas propiedades en consideración se puede decir que los operadores que cumplen estas propiedades no son demasiado destructivos ya que se enfocan en heredar en el mismo conjunto grandes subconjuntos de vértices que fueron asignados al mismo subconjunto en ambos padres. Finalmente el principio tras la tercera propiedad, es evitar el exceso de desbalanceo en los tamaños de los nuevos subconjuntos generados. Sin embargo, note que usualmente aparece cierto grado de desbalanceo, por lo que es necesario un método adicional para manejar correctamente el desbalanceo de los subconjuntos.

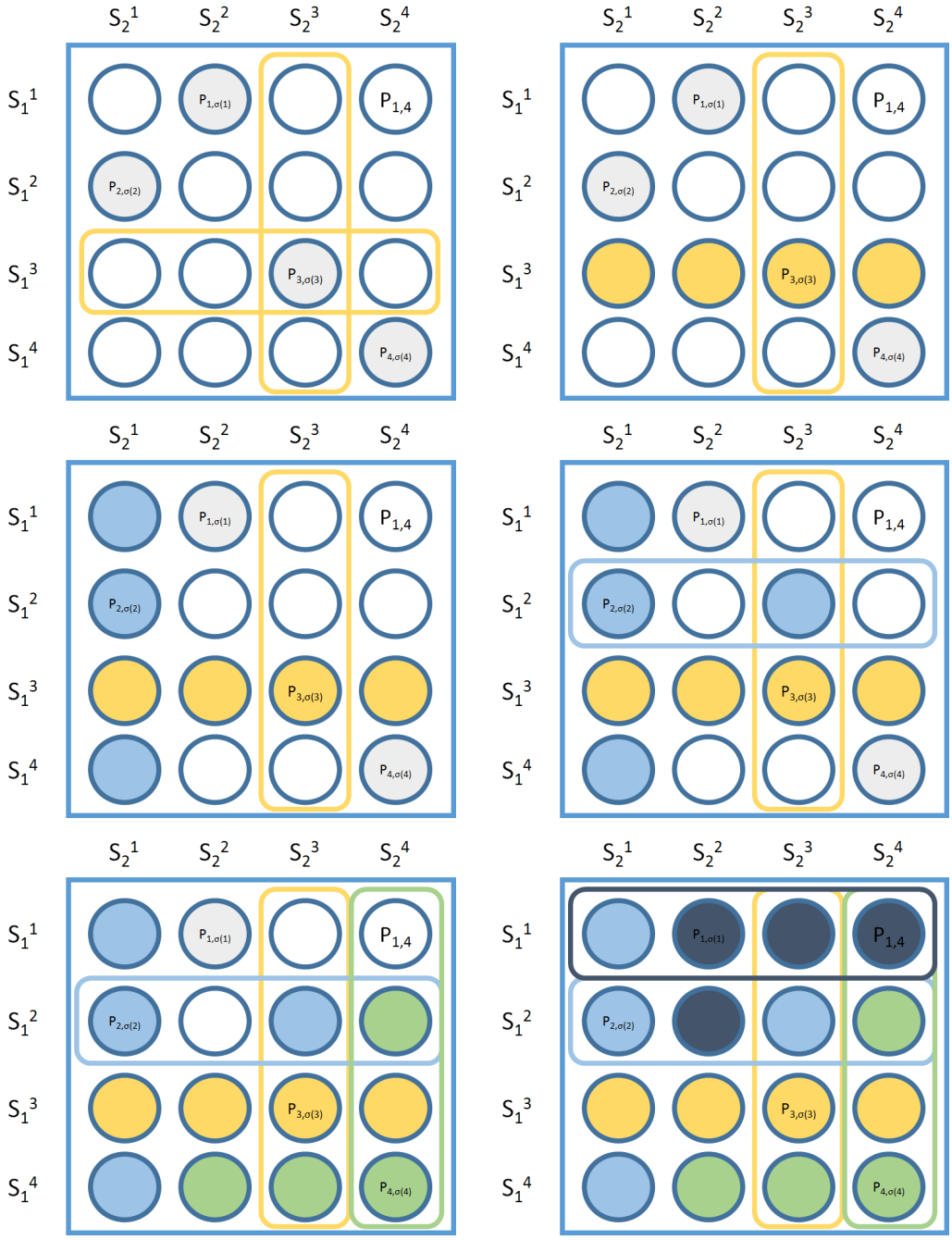

Figura 1: Ilustración del HBX

Construcción

Esta sección está dedicada a presentar el algoritmo para construir dos descendientes que cumplan las propiedades anteriormente mencionadas. Sea P la matriz de intersección previamente definida. Entonces el Algoritmo 2 es el proceso requerido para crear el primer individuo. Este proceso se ilustra en la Figura 1. La base del método es seleccionar en cada paso la posición $(i, \sigma(i))$ de la matriz, donde la unión de algunos elementos de la fila y columna correspondiente (ver detalles en línea 11 y 22) es maximizada. Nótese que se puede probar que en la línea 11 existen exactamente $k - j/2$ $P_{i,l}$ entradas no borradas de P y $j/2$ $P_{\sigma^{-1}(l),m}$ entradas tal que la columna m está "bloqueada". Algo similar ocurre en la línea 22 y esto prueba la propiedad 3 del HBX.

Para generar el segundo individuo, se intercambia el orden en el cual las filas y las columnas son consideradas, es decir, en las iteraciones pares se toman en consideración las columnas, mientras que en las iteraciones impares se toman en cuenta las filas. Nótese que independientemente del orden usado, ambos individuos comparten las mismas propiedades descritas anteriormente.

En la Figura 1 se muestra el funcionamiento del HBX para un individuo específico. En este caso $k = 4$, $\sigma = (2, 1, 3, 4)$. Cada una de las seis imágenes en la Figura 1 puede ser considerada como una matriz, P. Cada entrada $P_{i,j}$ de esas matrices representa la intersección de los conjuntos correspondientes en los padres S_i^1, S_j^2. En la última imagen, la unión de los conjuntos con el mismo color representa un elemento en la nueva partición, T. La segunda imagen muestra que T_1 es escogido de manera aleatoria como una fila de P, e ilustra que la columna 3 está bloqueada. Note que la fila fue elegida aleatoriamente y la columna considera la permutación dada por el Algoritmo Húngaro. En la tercera imagen, parte de T_2 es tomada de una columna de P y el resto de los elementos se toman de los conjuntos en la fila 2 que están en una columna bloqueada ($P_{2,3}$) tal y como se aprecia en la imagen 1. Similarmente T_3 y T_4 son creados en la imagen 5 y 6, todo siguiendo el procedimiento descrito en el Algoritmo 2.

Otro hecho importante es que para el nuevo operador, no se usa ninguna información propia de la definición del GPP. Por tanto, este

Algoritmo 3 Mutación basada en mover Componentes Conexas Aleatoriamente

```
1: Sea u un nodo elegido al azar.
2: Sea set un conjunto que inicialmente sólo contiene a u.
3: k = n_iter
4: while k > 0 do
5:     for u en set do
6:         for v vecino de u do
7:             Tomar p aleatorio en [0, 1].
8:             if p < p_m then
9:                 Agregar v a set
10:            end if
11:        end for
12:    end for
13:    k–
14: end while
15: Mover todos los vértices en set a una componente elegida aleatoriamente.
```

operador de cruce puede ser usado en cualquier problema de optimización en el cual compartir valores en diferentes variables sea una característica importante.

Operador de mutación

En los AEs es común incluir un operador de mutación, que en algunos casos se usa para realizar pequeños cambios que permitan intensificar, mientras que en otros realiza cambios más bruscos para escapar de óptimos locales. Dado que en nuestra propuesta se incluye búsqueda tabú para intensificar, se integró con una mutación disruptiva.

La mutación propuesta se basa en que en el GPP se desea que componentes conexas estén juntas en uno de los subconjuntos de la partición. La razón es que las componentes conexas encapsulan aristas comunes, implicando que se inducirán menos aristas de corte. Considerando esta propiedad, el principio de la mutación es mover partes de componentes conexas a uno de los subconjuntos. Se aplica el enfoque presentado en [24] (ver Algoritmo 3) para la selección de esta subcomponente. Luego, los vértices seleccionados se mueven a un conjunto S_i, donde i es seleccionado de manera aleatoria. En este trabajo se usa $p_m = 0{,}1$ y $n_{iter} = 5$. Es importante enfatizar que la mutación propuesta en [24] fue presentada para el problema de asignación de frecuencias.

Fase de reemplazamiento

Nuestro algoritmo memético aplica una fase de reemplazamiento original llamada "Estrategia de supervivencia del Mejor No Penalizado" ("Best-Non-Penalized (BNP) survivor selection strategy") (ver Algoritmo 4). Uno de los principios de la estrategia BNP es evitar la selección de individuos muy cercanos. Específicamente el enfoque intenta evitar la selección de pares de individuos que estén más cerca que un valor D uno del otro. El valor de D varía durante la ejecución, puesto que en las fases iniciales es importante explorar, mientras que en las últimas fases el proceso debe intensificar. En particular, la variable D es inicializada con base en el contenido de la primera población (ver línea 9) y disminuye de manera lineal hasta alcanzar el valor 0 al final de la ejecución.

En cada ejecución del operador de supervivencia se seleccionan N individuos de la población previa y los descendientes para sobrevivir. Iterativamente BNP selecciona el mejor elemento que no está penalizado. Luego, los individuos restantes con disimilitud a lo más D de los previamente seleccionados son penalizados. Si no es posible seleccionar un individuo no penalizado, se toma el individuo con mayor disimilitud a los actualmente seleccionados.

Un hecho importante que debe notarse es que el operador de supervivencia requiere una función de disimilitud entre los individuos. En este caso, se usa una función basada en el Algoritmo Húngaro. Sea τ una permutación de k elementos, entonces $d(S^1, S^2, \tau) = |V| - \sum_{i=1}^{k} \|S_i^1 \cap S_{\tau(i)}^2\|$. Nuestra disimilitud, $d(S^1, S^2)$ es definida como el mínimo de $d(S^1, S^2, \tau)$ sobre todas las permutaciones τ's. Este valor puede calcularse eficientemente usando el Algoritmo Húngaro.

Algoritmo 4 Técnica BNP de supervivencia

```
Require: Population, Offspring
1: for all I ∈ Offspring do
2:     I.cost = coste de corte asociado al individuo I
3: end for
4: Penalizados = ∅
5: CurrentIndividuals = Population ∪ Offspring
6: Best = Individuo con el coste más bajo de corte en CurrentIndividuals
7: NewPop = { Best }
8: CurrentIndividuals = CurrentIndividuals \ { Best }
9: Actualiza D tomando en cuenta el valor del tiempo actual (T_e), tiempo de paro
   (T_s) y valor inicial de D (D_I). Por ejemplo, en el caso de decremento lineal, realiza
   D = D_I − D_I * T_e/T_s. Tomamos D_I = distInitFactor * M, donde M es la disimilitud
   media en la primera población y distInitFactor ∈ [0, 1].
10: while (|NewPop| < N) do
11:     for all I ∈ CurrentIndividuals do
12:         I.DCN = disimilitud al individuo más cercano en NewPop
13:         if I.DCN < D then
14:             Penalized = Penalized ∪ {I}
15:             CurrentIndividuals = CurrentIndividuals \ {I}
16:         end if
17:     end for
18:     for all I ∈ Penalized do
19:         I.DCN = disimilitud al individuo más cercano en NewPop
20:     end for
21:     if CurrentIndividuals is empty then
22:         Selected = I con el I.DCN más grande en Penalized
23:         Penalized = Penalized \ {Selected}
24:     else
25:         Selected = I con el I.cost más bajo en CurrentIndividuals
26:         CurrentIndividuals = CurrentIndividuals \ {Selected}
27:     end if
28:     NewPop = NewPop ∪ {Selected}
29: end while
30: Population = NewPop
```

Algoritmo 5 Búsqueda Tabú

```
Require: I
1: it = 0
2: Best_I = I
3: while No se cumpla criterio de paro do
4:     (u, c) = movimiento válido de menor coste fuera de la lista de Tabú.
5:     I = I(u, c)
6:     if cost(Best_I) > cost(I) then
7:         Best_I = I
8:     end if
9:     it = it + 1
10:    Agregar (u, c) a lista de Tabú por t(it) iteraciones.
11: end while
12: I = Best_I.
```

Aprendizaje individual

Con la intención de crear individuos con mejor aptitud, en los algoritmos meméticos se incorpora un operador de "aprendizaje individual". Esta rutina se ejecuta después del balanceo y consiste de dos fases. La primera es una Búsqueda Tabú propuesta en [9] y la segunda es la estrategia denominada "búsqueda local perfectamente balanceada por medio de ciclos negativos" que se propuso en [21].

Galinier describe que esta variante de Búsqueda Tabú es una modificación del TabuCol propuesto para el coloreado de grafos [12]. Se recibe $I = \{V_1, V_2, \ldots, V_k\}$, una partición balanceada, y en cada iteración se mueve un nodo de un conjunto de la partición a otro. El movimiento del nodo u al conjunto c se considera válido si y sólo si el subconjunto en el que está el nodo u no es c y además $|V_c| < lim + \delta$, estableciendo $\delta = 1$ si $n = 0 \ (mod\ k)$ y $\delta = 0$, en caso contrario. Esta elección de δ es para asegurar que siempre exista al menos un movimiento válido.

Llamemos $I(u, c)$ a la partición cuya diferencia con I es que el nodo u es movido al conjunto c de la partición y $M(u, c)$ a dicho movimiento. El coste de $M(u, c)$ es igual a $cost(I(u, c)) - cost(I)$. El algoritmo 5 describe la búsqueda tabú. El criterio de paro puede ser la cantidad de segundos o iteraciones que queremos que se ejecute. En

nuestro caso el algoritmo termina cuando no se ha mejorado la mejor solución, $Best\_I$, en las últimas $noImprove$ iteraciones. El valor de $noImprove$ se eligió empíricamente y se muestra el proceso para hacerlo en la sección de validación experimental. La función $t(it)$, que calcula el número de iteraciones por las que un movimiento quedará en la lista tabú, es una función escalón periódica dependiente de un parámetro $maxT$. Para su construcción definiremos unos x_i que cumplirán que entre x_i y x_{i+1} la función será constante con cierto valor a_i. Formalmente la construcción de t se lleva a cabo de la siguiente forma:

- $b = \frac{1}{8}(1, 2, 1, 4, 1, 2, 1, 8, 1, 2, 1, 4, 1, 2, 1)$.
- $a_i = maxT \times b_i$.
- $x_1 = 1$ y $x_{i+1} = x_i + 4 \times maxT \times b_i$, para los i tales que b_i está definido.
- Finalmente $\forall it \in x_i, \ldots, x_{i+1}$, $t(it) = a_i$.

Se completa t para el resto de los naturales de manera periódica. Como ejemplo Galinier y Countiho usan $maxT = 200$, valor que adoptamos en nuestro esquema.

La intención con nuestro esquema es poder resolver el GPP en grafos de tamaños muy grandes. Esto provoca que se necesite tener una implementación eficiente del algoritmo 5. Al hacer una implementación trivial del algoritmo anterior, la parte donde se pierde la eficiencia en dicho código es en la línea 4, es decir, lo más costoso es encontrar el movimiento válido de menor coste fuera de la lista de Tabú, como suele ser habitual en las implementaciones de búsqueda tabú. Se pueden proponer varias estrategias para la implementación de este proceso. En la forma trivial se realiza una búsqueda lineal sobre todos los nodos y todos los conjuntos, lo que haría que esa línea tuviera costo al menos $O(n \times k)$, lo que es muy ineficiente para nuestros propósitos. La solución incorporada en nuestro esquema hace uso de una estructura llamada montículo binario actualizable (MBA). Cabe mencionar que la optimización aquí presentada es idea propia, pues Galinier no hace explícita su implementación. En esta estructura se almacenan pares (id, v), donde id es el identificador único de un elemento y v un valor de prioridad y la estructura permite encontrar el par (id, v) con menor v, así como actualizar $(id, v) \rightarrow (id, v')$ en tiempo logarítmico sobre el número de elementos en el montículo. En particular, nuestra implementación del MBA puede realizar las siguientes operaciones en tiempo logarítmico sobre el número de pares almacenados:

- **top()**: Devuelve el par (id, v) contenido en la estructura con menor valor v.
- **sum(id, w)**: Transforma el elemento (id, v) contenido en la estructura en el elemento $(id, v + w)$.
- **update(id, w)**: Cambia el elemento (id, v) en la estructura por (id, w).

Además, se tiene desarrollada la función $gainValue(u, i)$ que devuelve la ganancia que habría si se moviera el nodo u al conjunto i en el individuo actual. Si se mantiene una matriz de tamaño $n \times k$ donde se indique en la entrada $[u, i]$ cuántos nodos vecinos tiene u en el conjunto i, entonces esta operación se puede responder en $O(1)$ ya que el coste de mover el nodo u del conjunto i al j se puede calcular como la diferencia de las entradas $[u, i]$ y $[u, j]$. Con esto, la nueva manera de implementar la línea 4 se detalla en el Algoritmo 6.

Esto permite que cada iteración conlleve $O(NumVecinos(u) \times k \times log(\|V\|))$ que, al usar grafos ralos, mejora el rendimiento significativamente. En este caso se trabaja con k $MBAs$ debido a que necesitamos diferenciar entre los movimientos a diferentes conjuntos. En específico, si se usara sólo uno podría suceder que el mejor movimiento esté asociado a un conjunto ya lleno por lo que sería inválido y en dicho caso buscar entre los movimientos válidos se degeneraría a una búsqueda lineal.

Algoritmo 6 Búsqueda Tabú Eficiente

Require: I
1: $it = 0$
2: $Best\_I = I$
3: Inicializar $mba[k]$ donde cada $mba[c]$ es un MBA con la información $(u, cost(I(u, c)) - cost(I))$.
4: **while** No se cumpla criterio de paro **do**
5: Despenalizar cada uno de los elementos (u, c) que salieron de la lista de tabú. $mba[c].sum(u, -INF)$.
6: (u, c) = mín$_{i=1,2,\ldots,k}\{mba[i].top() \mid \|I_i\| < lim\}$
7: $I = I(u, c)$
8: **if** $cost(Best\_I) > cost(I)$ **then**
9: $Best\_I = I$
10: **end if**
11: $it = it + 1$
12: **for** $v \in \{vecinos\ de\ u\} \cup \{u\}$ y $i \in \{1, 2, \ldots, k\}$ **do**
13: Actualizar valor: $mba[i].update(v, gainValue(v, i))$.
14: **end for**
15: Agregar (u, c) a lista de Tabú por $t(it)$ iteraciones.
16: Penalizar: $mba[c].sum(u, INF)$.
17: **end while**
18: $I = Best\_I$.

Algoritmo 7 Operador de Balanceo

1: Sea I el individuo de entrada.
2: // Comienzo de la primera fase.
3: $t = n_{iterBal}$
4: **while** $t > 0$ y I no esté balanceado **do**
5: Seleccionar i aleatoriamente en $\{i$ tal que $\|I_i\|$ no es el conjunto más grande de $I\}$.
6: Encontrar el nodo u que al mover a I_i de máxima ganancia en $\cup I_j$ tal que $\|I_j\| > \|I_i\|$ y exista arista (u, v) con $v \in I_i$.
7: Mover u a I_i.
8: $t--$
9: **end while**
10: // Fin de la primera fase y comienzo de la segunda.
11: **while** ind no esté balanceado **do**
12: Elegir $u \in \cup I_j$ tal que $\|I_j\| > I_{max}$.
13: Mover u a un conjunto aleatorio I_k con $\|I_k\| < I_{max}$.
14: **end while**
15: // Fin de la segunda fase.

Balanceo

Esta sección está dedicada a presentar el operador de balanceo que se integró en nuestro esquema así como a detallar una implementación eficiente del mismo.

La rutina de balanceo consiste en dos fases y es muy importante debido a la restricción de balance dada por la definición del problema. Esta restricción puede no cumplirse después de la aplicación del operador de cruce y/o el operador de mutación por lo que se debe restaurar. En el Algoritmo 7 se da el procedimiento que está a cargo de esta parte.

En la primera fase, se mueven los nodos de los conjuntos más grandes a conjuntos más pequeños tomando en cuenta la implicación de moverlos en relación con el valor de la función de costo. Este proceso se repite $n_{iterBal}$ veces, el cual se fijó a $|V|$ en nuestra validación experimental.

La segunda fase, que provoca el cumplimiento de la restricción de desbalanceo, mueve de manera aleatoria nodos de conjuntos que no satisfacen la restricción dada a conjuntos que aún tienen capacidad suficiente para integrar más nodos.

Como en el caso de la Búsqueda Tabú, nos encontramos en una situación donde cierta parte del algoritmo necesita ser implementada de manera eficiente. En este caso es la línea 6 en el Algoritmo 7. Una implementación directa de esta línea puede tener costo del orden $O(\|E\|)$, lo que volvería este operador demasiado costoso computacionalmente. La idea para solucionar esto será nuevamente hacer uso del Montículo Binario Actualizable (MBA). Esta vez se mantendrán k^2 MBAs que contendrán información de la forma (e, v) donde $e \in E$ y v es un valor de prioridad. Específicamente se mantendrá una matriz de MBAs donde $(u, v) \in E$ estará en el MBA de la posición (i, j) si

Algoritmo 8 Operador de Balanceo Eficiente

1: Sea I el individuo de entrada.
2: $t = n_{iterBal}$.
3: Construir $mba[k][k]$ con información actualizada.
4: **while** $t > 0$ y I no esté balanceado **do**
5: Seleccionar i aleatoriamente en $\{i$ tal que $\|I_i\|$ no es el conjunto más grande de $I\}$.
6: Hallar $(u, v) = \text{mín}_{j \in 1,2,\ldots,k}\{mba[j][i].top() \mid \|I_j\| > \|I_i\|\}$.
7: Eliminar las aristas vecinas de u de sus respectivos mba.
8: Mover u a I_i.
9: Insertar las aristas vecinas de u en sus nuevos mba con el nuevo valor de reducción en aptitud.
10: $t - -$
11: **end while**
12: **while** I no esté balanceado **do**
13: Elegir $u \in \cup I_j$ tal que $\|I_j\| > I_{max}$.
14: Mover u a un conjunto aleatorio I_k con $\|I_k\| < \|I_{max}\|$.
15: **end while**

y sólo si $(I[u], I[v]) = (i, j)$ y su valor de prioridad asociado será la reducción del valor de aptitud si se moviera el nodo u al conjunto $I[v]$. Además, llamaremos una arista, (u, v), vecina de un nodo, w, si u o v es vecino de w. Bajo estas definiciones, algo que es importante notar es que, si se mueve el nodo u al conjunto $I[v]$, las únicas aristas que cambian su valor de prioridad son las aristas vecinas de u. Si el grafo es ralo, este valor, $eVec(u)$, no será demasiado grande. Con esto en mente, el operador de balanceo queda ilustrado en el Algoritmo 8.

El nuevo algoritmo vuelve a la sección equivalente a la línea 5 en un procedimiento con costo $O((k + eVec(u)) \times log(\|E\|))$, que es mucho mejor que el anterior costo. Lo anterior muestra una vez más que se puede aprovechar el uso de estructuras de datos más complejas para tener implementaciones eficientes de un algoritmo.

Propuesta final

Una vez revisada cada una de las componentes necesarias para la implementación de un algoritmo memético, la versión final de nuestra propuesta se ve reflejada en el Algoritmo 9. Debido a las componentes de nuestro algoritmo, lo decidimos llamar "Algoritmo Memético con Cruce basado en el Algoritmo Húngaro y Control de Diversidad" y en adelante lo abreviaremos MAHMBCDP (Memetic Algorithm with Hungarian Matching Based Crossover and Diversity Preservation). En este algoritmo se comienza con una población aleatoria con la intención de muestrear soluciones distantes y se evalúan las soluciones generadas. Recordemos que la disimilitud inicial del operador de supervivencia se fija con base en las disimilitudes iniciales de la población. Por esta razón se realiza primero una fase de aprendizaje individual, pues calcular las disimilitudes antes del aprendizaje haría que dicha disimilitud fuera demasiado grande y se forzaría el mantenimiento de soluciones de muy baja calidad. De ahí se entra a un ciclo del cual no se saldrá hasta que haya transcurrido un tiempo T. Dentro del ciclo se inicia suponiendo que todos los individuos tienen su valor de aptitud disponible durante la selección de los padres de la siguiente generación. La selección de dichos padres se hace por medio de torneo binario, un método que necesita comparar los valores de aptitud de los individuos. Una vez elegidas las parejas, si fueron seleccionadas para el cruce, se llama al operador de cruce basado en el Algoritmo Húngaro. A continuación se decide si el individuo será además mutado. Al salir de esta última rutina tendremos nuevos individuos cuyo valor de aptitud puede no estar actualizado y pueden no cumplir la restricción de balance. Por esta razón, inmediatamente se lanza el método de reparación, el cual consiste en el rebalanceo previamente explicado. Una vez balanceadas se llama nuevamente al operador de aprendizaje individual con la intención de mejorar a los individuos que competirán por continuar en la población. Esta fase incluye a la búsqueda tabú y la optimización basada en detección de ciclos negativos ya mencionada. Finalmente se llama al operador de supervivencia eligiendo así a los individuos que conformarán la siguiente generación del algoritmo.

Algoritmo 9 Algoritmo Memético con Cruce basado en el Algoritmo Húngaro y Control de Diversidad

1: **Inicialización**: Genera una población inicial P_0 con N individuos. Asignar $t = 0$.
2: **Evaluación**: Evaluar todos los individuos en la población.
3: **Balanceo**: Los individuos se balancean inicialmente.
4: **Búsqueda Tabú**
5: **Búsqueda perfectamente balanceada por medio de ciclos negativos**
6: **Inicializar** D_I: Se hace $D_I = distInitFactor * M$, donde M es la disimilitud media de la población.
7: **while** Tiempo transcurrido sea menor que T **do**
8: **Emparejamiento**: Realizar un torneo binario sobre P_t para formar $\frac{N}{2}$ parejas de padres.
9: **Cruce Basado en el Algoritmo Húngaro**: Se aplica sobre las parejas formadas con probabilidad p_c para crear el conjunto de hijos CP.
10: **Mutación**: Se mutan los individuos con probabilidad p_m.
11: **Evaluación**: Evaluar todos los individuos en la población.
12: **Balanceo**: Aplicar la fase de balanceo para cumplir las restricciones de balance.
13: **Búsqueda Tabú**
14: **Búsqueda perfectamente balanceada por medio de ciclos negativos**
15: **BNP**: Se aplica a la combinación de P_t y CP y guarda en P_{t+1} el resultado.
16: $t = t + 1$
17: **end while**

Validación experimental

En esta sección se describe el trabajo experimental llevado a cabo para validar las propuestas algorítmicas de la tesis en la que este trabajo se basa. La sección incluye varios experimentos enfocados a analizar y comprender mejor el rendimiento y comportamiento de la técnica desarrollada y la importancia de las componentes propuestas. Además se incluyen comparativas con las mejores soluciones reportadas hasta la fecha para mostrar la contribución de la técnica en lo referente a obtención de particiones de muy alta calidad.

Descripción general de los experimentos

Con el fin de validar el rendimiento de las diferentes componentes propuestas, especialmente del mecanismo de gestión de diversidad, se implementaron varias componentes diferentes y se realizaron comparativas exhaustivas entre ellas. Los esquemas fueron implementados usando la herramienta METCO [14] y teniendo en cuenta que nuestra propuesta es un algoritmo estocástico, cada ejecución se repitió 30 veces y la comparación entre los diferentes esquemas se llevó a cabo aplicando un conjunto de tests estadísticos [22]. Para evaluar el desempeño de nuestra propuesta, se usaron instancias del TGPA de Chris Walshaw [1]. Este es un sitio que ha sido mantenido desde el año 2000 e incluye resultados sobre los paquetes de particionado más importantes. Para cada uno de estos grafos, la mejor partición conocida para $k \in \{2, 4, 8, 16, 32, 64\}$ y $\epsilon \in \{0.0, 0.01, 0.03, 0.05\}$ se encuentra disponible. Para nuestra validación se tomaron en cuenta los valores $k \in \{4, 8, 16, 32, 64\}$ y $\epsilon = 0{,}0$ para probar nuestro algoritmo.

Primer experimento: selección de parámetros

En primer lugar, como es habitual en el campo de optimización con metaheurísticas, el enfoque desarrollado tiene un conjunto de parámetros para los que no es trivial fijar un valor. Considerando el caso con reemplazamiento BNP los parámetros son el tamaño de población, la probabilidad de cruce, la probabilidad de mutación, el valor de $distInitFactor$ para fijar la disimilitud inicial de penalización y $noImprove$ para la búsqueda tabú. El tamaño de población se fijó a 50, la probabilidad de cruce (p_c) a 0.85 y la probabilidad de mutación (p_m) a 0.1. Estos valores son cercanos a los estándar y estamos conscientes de que se podrían obtener algunas mejoras aplicando esquemas de calibración de parámetros [15]. Sin embargo, puesto que se realizaron ejecuciones muy largas (el criterio de paro se fijó a 48 horas) y estas mostraron resultados de gran calidad, no se llevó a cabo más experimentación con dichos parámetros.

Dado que una de las características más importantes de nuestro enfoque es que considera la diversidad explícitamente y que esta de-

Tabla 1: Resultados del MAHMBCDP para k = 16, 64 con $noImprove = 5000$

add20	$k = 16$				$k = 64$			
DInit	Mejor	Media	Mediana	Peor	Mejor	Media	Mediana	Peor
0.0	2044	2064.5	2062.5	2091	2953	2982	2972	3025
0.2	2040	2051.7	2052	2064	2943	2950.6	2949	2959
0.4	2040	2044.8	2043	2052	2941	2949.5	2949.5	2957
0.6	2040	2042.1	2042.5	2046	2943	2950.6	2951	2959
0.8	2040	2041.7	2042	2045	2946	2953.1	2953	2965
1.0	2040	2042.5	2043	2045	2946	2952.5	2951.5	2965
whitaker3	$k = 16$				$k = 64$			
DInit	Mejor	Media	Mediana	Peor	Mejor	Media	Mediana	Peor
0.0	1088	1091.2	1091	1098	2498	2512.2	2511	2526
0.2	1085	1087	1088	1088	2486	2495.4	2494.5	2511
0.4	1085	1085.8	1085	1088	2484	2489	2489	2495
0.6	1085	1085.7	1085	1088	2484	2489.9	2487.5	2508
0.8	1085	1086.2	1085	1088	2484	2487.4	2487	2495
1.0	1085	1086.3	1085	1088	2484	2486.7	2487	2491
bcsstk29	$k = 16$				$k = 64$			
DInit	Mejor	Media	Mediana	Peor	Mejor	Media	Mediana	Peor
0.0	21941	22233.9	22204.5	22800	55591	55930.1	55890.5	56761
0.2	21900	21919.8	21900.5	22009	55591	55930.1	55890.5	56761
0.4	21900	21927.5	21920	21991	55356	55637.2	55634	55838
0.6	21900	21936.6	21927	22022	55332	55589.8	55588.5	55848
0.8	21900	21941.4	21944.5	22020	55345	55601.1	55629	55803
1.0	21902	21935.1	21918	22004	55338	55625.6	55608	55917

Tabla 2: Resultados del MAHMBCDP para k = 16, 64 con $distInitFactor = 0{,}4$

add20	$k = 16$				$k = 64$			
noImprove	Mejor	Media	Mediana	Peor	Mejor	Media	Mediana	Peor
200	2040	2047.5	2046	2056	2947	2953.9	2953	2964
2000	2040	2046	2045.5	2053	2943	2950.8	2950	2962
5000	2040	2044.1	2042	2054	2944	2950.2	2950	2962
10000	2040	2043.7	2042	2052	2944	2949.7	2950	2956
15000	2040	2043.2	2041.5	2052	2944	2949.6	2949	2955
25000	2040	2044.5	2043	2052	2943	2950.1	2951	2955
35000	2040	2043.8	2042	2052	2944	2950.2	2950	2961
50000	2040	2045.3	2043	2052	2944	2949.7	2950	2955
60000	2040	2044.5	2042	2053	2946	2950.7	2950	2960
75000	2040	2044.6	2043	2053	2942	2949.6	2949	2957
whitaker3	$k = 16$				$k = 64$			
noImprove	Mejor	Media	Mediana	Peor	Mejor	Media	Mediana	Peor
200	1085	1087.2	1088	1089	2485	2492.8	2492	2504
2000	1085	1085.8	1085	1088	2484	2489.1	2489	2498
5000	1085	1085.5	1085	1088	2484	2488.6	2487	2496
10000	1085	1085.5	1085	1088	2484	2488.4	2487.5	2499
15000	1085	1085.5	1085	1088	2483	2487.1	2487	2493
25000	1085	1086.1	1085	1088	2484	2486.9	2487	2493
35000	1085	1086.6	1087	1088	2484	2488.3	2488.5	2493
50000	1085	1086.7	1086.5	1088	2484	2487.7	2488	2492
60000	1085	1086.7	1087.5	1088	2484	2488.5	2489	2493
75000	1085	1087	1088	1088	2484	2488.6	2489	2493
bcsstk29	$k = 16$				$k = 64$			
noImprove	Mejor	Media	Mediana	Peor	Mejor	Media	Mediana	Peor
200	21900	21941.2	21946.5	22032	55500	55714.8	55695	56043
2000	21900	21920.3	21904	21989	55442	55657.4	55629.5	56006
5000	21900	21917.7	21909.5	21958	55407	55639.8	55618	55879
10000	21900	21927.8	21904	22059	55408	55631.5	55610.5	55905
15000	21900	21916	21903	21988	55314	55653.2	55651	55908
25000	21900	21929.6	21925	22017	55416	55708	55672.5	56010
35000	21900	21930.3	21917	21998	55367	55655.1	55671.5	55967
50000	21900	21920	21904	22019	55396	55636.6	55646.5	55905
60000	21900	21944.7	21933	22066	55371	55642.7	55625	55925
75000	21900	21919.3	21903	21984	55500	55671.3	55651	56011

pende de $distInitFactor \in [0, 1]$, se realizó un esfuerzo por buscar un valor adecuado para este parámetro. De la misma forma, como el valor $noImprove$ influye en gran medida sobre el número de generaciones evolucionadas, también se hizo el esfuerzo de realizar pruebas para fijar dicho valor. Para $distInitFactor$ se consideró que podía tomar uno de seis valores diferentes equidistribuidos en el intervalo $[0, 1]$, mientras que para $noImprove$ se consideraron los siguientes valores: $200,\ 2,000,\ 5,000,\ 10,000,\ 15,000,\ 25,000,\ 35,000,\ 50,000,\ 60,000,\ 75,000$. Hacer experimentos con todas las posibles combinaciones es muy costoso computacionalmente, por lo que se optó por optimizar fijando uno de los parámetros y reiterando hasta convergencia. Nótese que en este proceso se está optimizando cada parámetro de forma independiente al otro, a pesar de que existen dependencias entre ellos. Sin embargo, debido al alto costo computacional asociado se decidió realizar el proceso de esta forma. Además, estos experimentos solo se realizaron con tres grafos, add20, whitaker3 y bcsstk29 con $k = 16, 64$. Para elegir el parámetro en cada paso, se compararon todas las configuraciones ejecutadas, y se eligió aquella que maximizaba el número de veces que fue mejor estadísticamente menos el número de veces que fue peor estadísticamente.

Los parámetros iniciales se fijaron estableciendo $distInitFactor$ a 0,6 y $noImprove$ a $5,000$. En la Tabla 1 se muestran los resultados con $noImprove$ fijo para los diferentes valores de $distInitFactor$. Con estos experimentos se concluyó que el mejor valor para $distInitFactor$ en este caso era 0,4. Análogamente en la Tabla 2 se varía ahora $noImprove$ y los test estadísticos arrojaron que el mejor valor pa-

Tabla 3: Resultados del MAHMBCDP para k = 16, 64 con $noImprove = 15000$

add20	$k = 16$				$k = 64$			
DInit	Mejor	Media	Mediana	Peor	Mejor	Media	Mediana	Peor
0.0	2053	2065	2060.5	2094	2950	2984.2	2974.5	3026
0.2	2041	2050.8	2052	2059	2943	2949.8	2949.5	2958
0.4	2040	2043.8	2043	2053	2944	2949.5	2949	2958
0.6	2040	2041.2	2041	2043	2945	2951.7	2951.5	2959
0.8	2040	2041.5	2041	2044	2943	2950.9	2950	2961
1.0	2040	2041.9	2043	2045	2946	2952.9	2952.5	2959
whitaker3	$k = 16$				$k = 64$			
DInit	Mejor	Media	Mediana	Peor	Mejor	Media	Mediana	Peor
0.0	1085	1090.3	1090	1094	2492	2501.8	2502	2514
0.2	1085	1087.2	1088	1088	2484	2491	2490.5	2499
0.4	1085	1086	1085	1088	2484	2488	2488	2494
0.6	1085	1086.2	1085.5	1088	2484	2487.8	2488	2491
0.8	1085	1086.6	1086.5	1088	2484	2493.7	2488	2527
1.0	1085	1087	1088	1088	2484	2487.2	2488	2491
bcsstk29	$k = 16$				$k = 64$			
DInit	Mejor	Media	Mediana	Peor	Mejor	Media	Mediana	Peor
0.0	21973	22287.3	22265	22865	56052	56563.6	56499	57540
0.2	21900	21919.1	21901.5	22007	55582	55873.3	55861.5	56127
0.4	21900	21924.2	21904	21980	55434	55693.8	55699	55934
0.6	21900	21933.5	21932	22066	55405	55597.4	55585	55800
0.8	21900	21928.1	21907	21988	55343	55607.3	55608.5	55831
1.0	21900	21938.3	21936	22006	55424	55624.6	55627	56009

ra este parámetro en este caso era 15,000. Finalmente, en la última iteración al variar $distInitFactor$ este queda fijo otra vez en 0,4, por lo que el proceso finaliza. Los resultados se presentan en la Tabla 3. Con esto se terminó de decidir los parámetros que usamos para el resto de las ejecuciones. Así, $distInitFactor$ se fijó a 0,4 y $noImprove$ se fijó a 15,000.

Segundo experimento: análisis de diversidad

Una de las principales novedades de nuestro esquema es el control explícito de la diversidad. Esto se lleva a cabo en la fase de supervivencia con el operador de remplazamiento BNP. La Figura 2 muestra la evolución del fitness y la diversidad a lo largo de las ejecuciones y ofrecen una buena ilustración del tipo de casos que encontramos. El grafo que se está visualizando en este caso es el (*vibrobox*) para $k = 8, 64$. Notamos que en ambos casos se tiene un decremento paulatino de la diversidad, que es lo que se esperaba. Además, en las figuras relativas al valor de aptitud se indica también cuál es el mejor valor que se conocía antes de comenzar la tesis. Podemos ver que en el caso de $k = 8$ se supera dicho valor mientras que en el caso $k = 64$ no se alcanza. En el siguiente experimento se ofrecen resultados adicionales que comparan los resultados alcanzados en la tesis con los mejores conocidos antes de empezar la tesis.

Para validar el impacto que se tiene sobre la evolución de la diversidad y confirmar los beneficios del BNP se comparó nuestro operador con otros esquemas que han ganado gran popularidad, habiendo sido propuestos algunos de ellos como mecanismos para lidiar con la convergencia prematura. Específicamente, se obtuvieron resultados con MAHMBCDP, RTS, CD/RW, HGSADC y GEN_ELIT. La Figura 3 muestra la evolución de la diversidad y la aptitud de cada uno de los diferentes algoritmos al ser aplicados al grafo *whitaker3*. Para entender a fondo las razones de la superioridad del operador BNP hay que poner especial atención a este tipo de gráficas de diversidad. En las gráficas, la diversidad se toma como la media de la disimilitud entre cada par de individuos de la población. Es notorio que MAHMBCDP decrece la diversidad de manera paulatina durante toda la ejecución del algoritmo. Algo muy diferente ocurre en los otros esquemas que en algunos casos pierden la diversidad muy tempranamente y en otros mantienen alta diversidad en toda la ejecución. Al comenzar con una diversidad alta, en MAHMBCDP el espacio de soluciones es explorado de manera general. Esto también explica por qué, al principio de las ejecuciones, MAHMBCDP tiene una calidad en la aptitud inferior a los demás esquemas, pues no intensifica en cada zona sino que sólo trata de detectar zonas promisorias. Una vez que se han identificado zonas promisorias para el algoritmo, con el paso del tiempo y la reducción de la diversidad, se comienza a intensificar a mayor profundidad. Esto se ve reflejado en la calidad de las soluciones al superar a cada uno de los otros esquemas en las últimas horas.

Tercer experimento: comparativa con el estado del arte

Para ilustrar la efectividad del algoritmo memético propuesto, se realizaron pruebas con grafos adicionales del TGPA y se compararon con los mejores resultados encontrados hasta la fecha. Se probó el MAHMBCDP con los valores $k = \{4, 8, 16, 32, 64\}$. Las Tablas 4, 5, 6 resumen los resultados obtenidos con MAHMBCDP. Adicionalmente al mejor, la media y el peor resultado obtenido, la mejor solución conocida encontrada por cualquier esquema previo a nuestra tesis se muestra en la columna TGPA. En la tablas se muestra en negrita los casos en los que MAHMBCDP alcanzó el mejor resultado conocido. Además, aquellos casos en los que se superó al mejor conocido se marcan con un asterisco. Cabe destacar que de los 115 casos probados, la mejor solución conocida fue alcanzada o superada en 80 casos de ellos, mientras que se pudo superar en 35 casos.

Conclusiones y trabajos futuros

El problema del particionado de grafos es un problema clásico y muy estudiado en la comunidad de optimización combinatoria. Se ha estudiado usando una gran cantidad de técnicas de las cuales se realizó una revisión en nuestra tesis. En este trabajo se propuso un algoritmo memético con gestión de diversidad para el GPP cuyas novedades son un operador de cruce diseñado especialmente para particiones y una estrategia de reemplazamiento que ayuda a mitigar la convergencia prematura y toma en cuenta el criterio de paro fijado por el usuario. Se consideró que esta clase de algoritmos podrían mejorar el estado del arte del problema puesto que se ha visto su efectividad en problemas como el Problema del Coloreado de Grafos y hasta la fecha no se habían probado con el GPP. Además, dado que se pretendía ejecutar la propuesta durante miles de generaciones, se cuidó particularmente el aspecto de la eficiencia en las subrutinas implementadas. En lo referente a contribuciones específicas relativas a los resultados obtenidos, cabe destacar que una vez elegidos los operadores de manera ade-

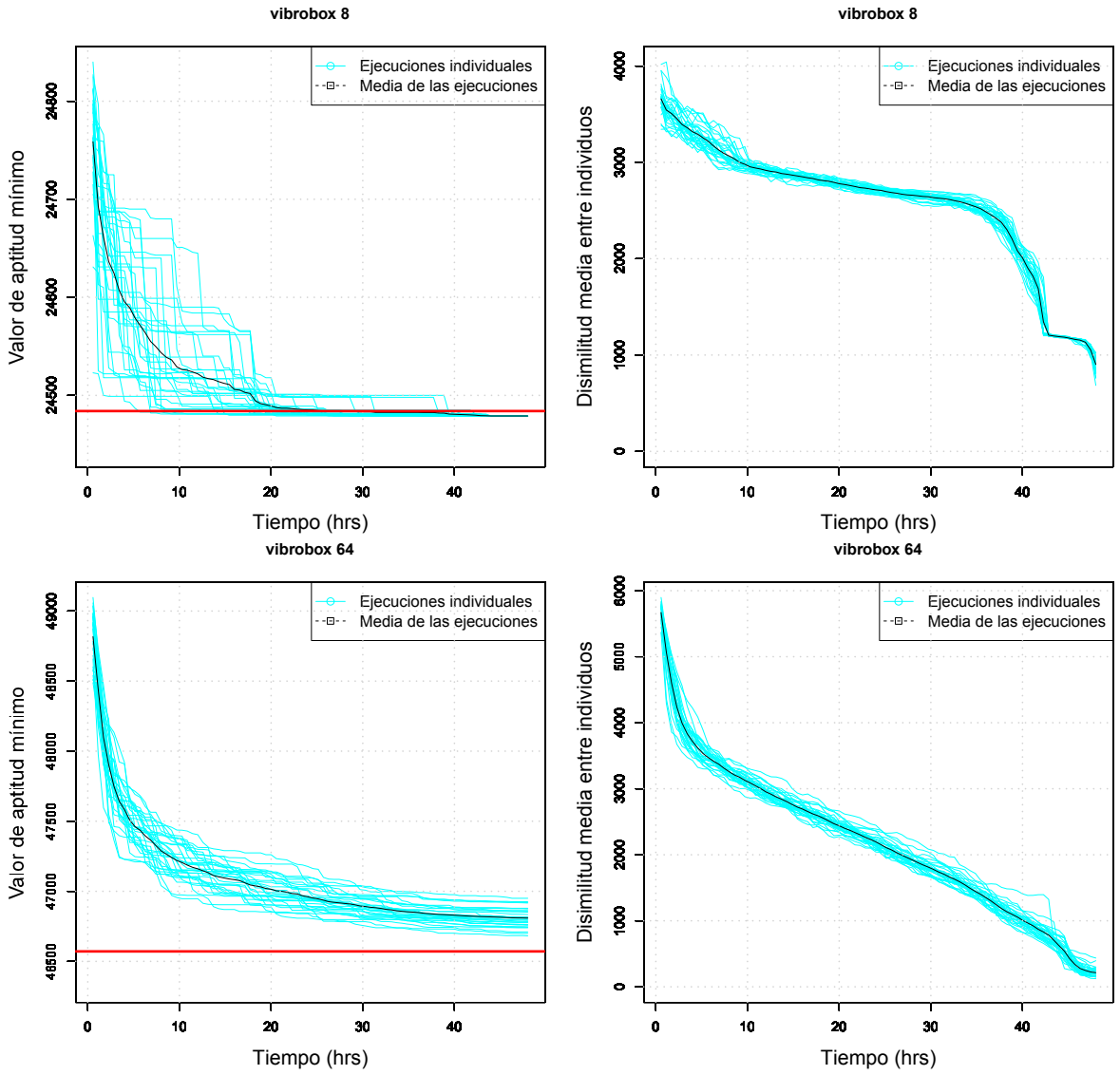

Figura 2: Evolución de la aptitud y la diversidad para instancias *vibrobox*, $k = 8, 64$ en el método MAHMBCDP

Tabla 4: Resultados del MAHMBCDP para k = 4, 8 con la parametrización final ($distInitFactor = 0{,}4$ y $noImprove = 15,000$)

	$k = 4$					$k = 8$				
Grafo	TGPA	Mejor	Media	Mediana	Peor	TGPA	Mejor	Media	Mediana	Peor
add20	1152	**1151***	1151	1151	1151	1681	**1678***	1680.5	1680.5	1684
data	382	**382**	382	382	382	668	**668**	668	668	668
3elt	201	**201**	201	201	201	345	**345**	345	345	345
uk	41	42	42.6	43	43	83	84	85.5	85	88
add32	34	**34**	34	34	34	67	**67**	67.6	68	68
bcsstk33	21717	**21717**	21717	21717	21717	34437	**34437**	34437	34437	34437
whitaker3	381	**381**	381	381	381	656	**656**	656	656	656
crack	366	**366**	366	366	366	679	**679**	679	679	679
wing_nodal	3575	**3575**	3575	3575	3575	5435	**5435**	5435.2	5435	5436
fe_4elt2	349	**349**	349	349	349	607	**606***	606.6	606.5	608
vibrobox	18976	**18976**	18976.5	18976	18978	24484	**24479***	24479	24479	24479
bcsstk29	8035	**8035**	8035	8035	8035	13975	**13958***	13958	13958	13959
4elt	326	**326**	326	326	326	545	**545**	545	545	545
fe_sphere	768	**768**	768	768	768	1156	**1156**	1156	1156	1156
cti	954	**954**	954	954	954	1788	**1788**	1788	1788	1788
memplus	9448	10646	10922.5	10918.5	11007	11712	12409	12914.2	12999.5	13590
cs4	932	972	1575.1	1610	2545	1440	1919	2615.8	2664	2974
bcsstk30	16651	**16651**	16686.6	16651	16972	34846	**34846**	34854.3	34846	34893
bcsstk31	7351	**7342***	7566	7531.5	7927	13283	**13272***	13583.1	13498	14217
fe_pwt	705	**705**	705	705	705	1447	**1447**	1447	1447	1447
bcsstk32	9311	**9311**	9433.3	9430	10447	20009	**19874***	20498.3	20232.5	22359
fe_body	599	619	1702.7	679	6150	1033	1061	1323	1153.5	2775
t60k	209	2224	5669.1	6876.5	8547	456	8114	10467	10888.5	11594

cuada se procedió a la validación experimental con las instancias del TGPA de Chris Walshaw y en las mismas se logró superar las mejores soluciones que había hasta la fecha en 35 instancias. Estos resultados fueron validados por Chris Walshaw y ahora nuestro algoritmo se encuentra listado en el TGPA como uno de los métodos a batir. Esto es un gran logro tomando en cuenta la cantidad de trabajo y

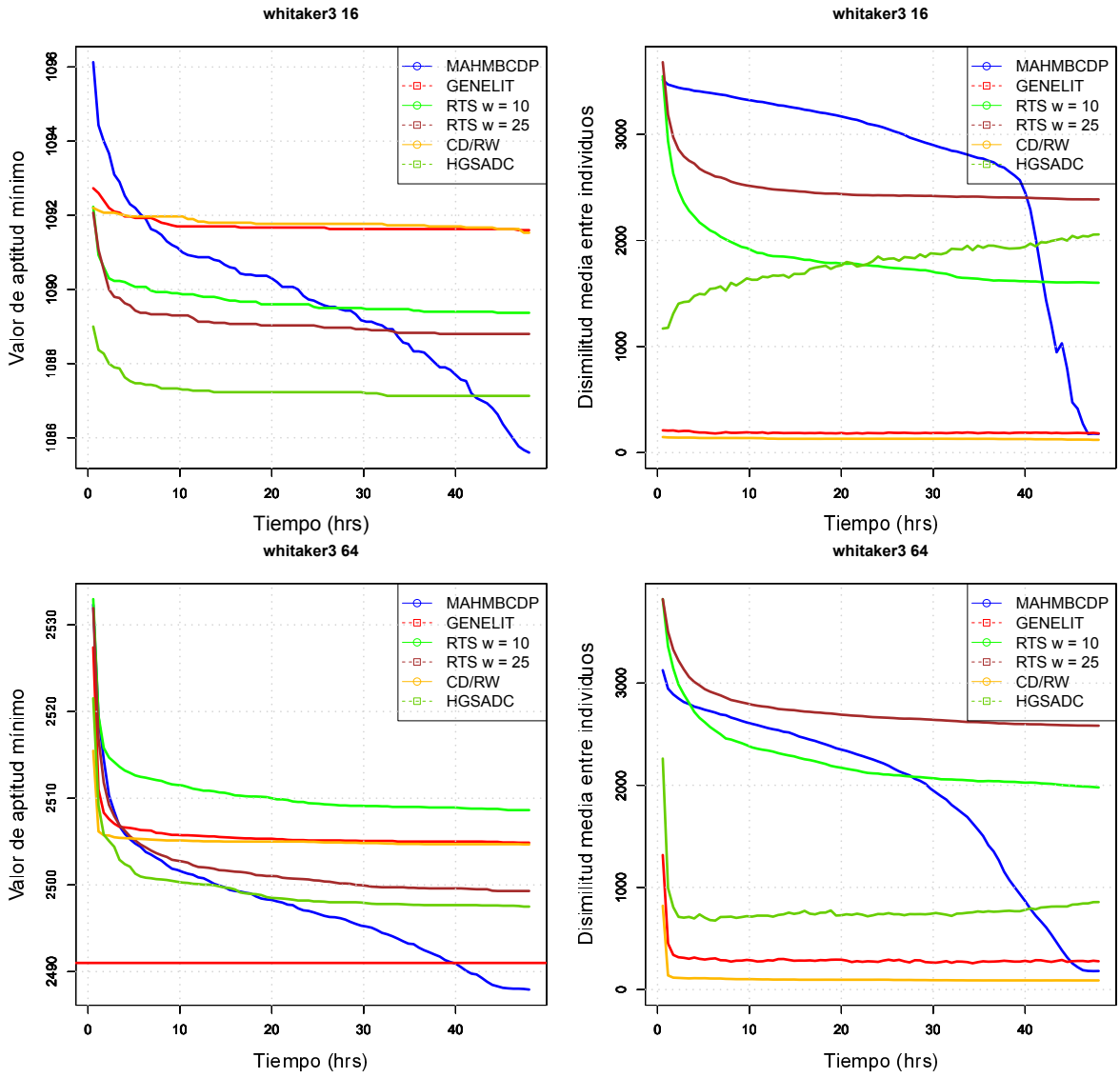

Figura 3: Evolución de la media de la aptitud mínima de treinta ejecuciones y la media de la diversidad para instancias *whitaker3*, $k = 16, 64$ y los diferentes operadores de supervivencia.

Tabla 5: Resultados del MAHMBCDP para k = 16, 32 con la parametrización final ($distInitFactor = 0{,}4$ y $noImprove = 15,000$)

	$k = 16$					$k = 32$				
Grafo	TGPA	Mejor	Media	Mediana	Peor	TGPA	Mejor	Media	Mediana	Peor
add20	2041	**2040***	2043.7	2043	2052	2361	**2357***	2359.8	2359	2365
data	1127	**1127**	1127	1127	1127	1799	**1799**	1799.1	1799	1800
3elt	573	**573**	573	573	573	960	**960**	960.2	960	961
uk	145	148	150.4	150	155	247	255	259.3	259.5	265
add32	118	119	125.9	126	132	213	**213**	223.5	224.5	234
bcsstk33	54680	**54680**	54680.5	54680	54685	77414	**77410***	77414.5	77415	77417
whitaker3	1085	**1085**	1085.9	1085	1088	1668	**1668**	1669.2	1669	1672
crack	1088	**1088**	1088	1088	1088	1679	**1678***	1680.4	1681	1682
wing_nodal	8334	**8331***	8333.2	8332	8342	11768	**11760***	11766.6	11765	11782
fe_4elt2	1007	1008	1008.1	1008	1009	1614	**1613***	1615	1615	1621
vibrobox	31850	32324	32759.7	32786.5	32898	39477	**39397***	39634.1	39630	40102
bcsstk29	21905	**21900***	21922.5	21910.5	21984	34737	**34721***	34837.9	34802	35051
4elt	934	**934**	934.9	934	940	1551	**1551**	1552.1	1552	1553
fe_sphere	1714	**1714**	1714	1714	1714	2488	**2488**	2489.4	2489	2490
cti	2793	**2792***	2792.8	2793	2794	4046	**4043***	4047.3	4045	4066
memplus	12895	14125	14271.3	14266.5	14582	13953	15338	15362.6	15362.5	15385
cs4	2075	2718	3065	3044	3435	2928	3472	3776.7	3778	4145
bcsstk30	70408	**70404***	70576.9	70443	71118	113336	**113013***	113697.6	113642	114619
bcsstk31	23869	**23605***	23910.1	23914	24262	37158	37363	37838.4	37887	38695
fe_pwt	2830	2835	2836.4	2836.5	2838	5575	**5575**	5583.4	5579	5605
bcsstk32	36250	**36141***	36744.9	36698	37658	60038	**59701***	60426.1	60389.5	61646
fe_body	1736	1823	2074.1	1962.5	3287	2846	2929	2996.1	2971	3472
t60k	813	9684	11317.1	11463	12379	1323	3693	8952.7	9315	9845

publicaciones que se han usado para lidiar con el GPP y en particular con las instancias en el TGPA. Como principal desventaja de nuestra propuesta podemos mencionar que incorpora operadores muy costosos. Esto provoca que al aumentar el número de nodos o el número

Tabla 6: Resultados del MAHMBCDP para k = 64 con la parametrización final ($distInitFactor = 0{,}4$ y $noImprove = 15,000$)

	$k = 64$				
Grafo	TGPA	Mejor	Media	Mediana	Peor
add20	2949	**2944***	2949.4	2949	2955
data	2839	**2835***	2838.7	2838.5	2847
3elt	1532	**1532**	1533.5	1533.5	1535
uk	408	416	420.9	420.5	427
add32	485	**485**	495	493	517
bcsstk33	107185	**107179***	107268.1	107272	107353
whitaker3	2491	**2484***	2487.7	2488	2492
crack	2535	2538	2541.2	2541	2547
wing_nodal	15775	**15769***	15773.4	15772.5	15780
fe_4elt2	2478	**2471***	2478.2	2478.5	2485
vibrobox	46571	46682	46808.9	46807.5	46950
bcsstk29	55241	55364	55644.7	55661.5	55856
4elt	2565	**2562***	2565.8	2566	2569
fe_sphere	3543	**3543**	3545.2	3545.5	3547
cti	5629	**5618***	5646.4	5648	5667
memplus	16223	17204	17295	17281	17420
cs4	4027	4332	4490.9	4481	4702
bcsstk30	171153	**171088***	171801.7	171772	172624
bcsstk31	57402	**57347***	57539.9	57500	57996
fe_pwt	8180	8275	8317.7	8313.5	8359
bcsstk32	90895	91109	92042	92111	92756
fe_body	4730	4798	4885.5	4895	4981
t60k	2077	2311	2856.8	2841.5	3450

de conjuntos en las particiones no se evolucionen demasiadas generaciones, algo muy importante en los esquemas basados en control de diversidad y en particular en el nuestro, por lo que en este aspecto hay que seguir mejorando el esquema.

Existen muchas lineas de investigación que aún pueden ser exploradas. En primer lugar, consideramos que una línea promisoria de trabajo es rediseñar el operador de mutación ya que algunas pruebas preliminares realizadas al final de la tesis mostraron que al incorporar una mutación de mayor intensidad, la calidad de las soluciones se puede mejorar significativamente en algunos tipos de grafos. También, para continuar con la validación de nuestro esquema, se podría comparar contra otros operadores de cruce más sofisticados. Otra línea podría consistir en rediseñar el operador de reemplazamiento utilizando control adaptativo, pues a pesar de que se logró que la diversidad decreciera poco a poco durante las ejecuciones, no se logró inducir un decremento lineal tan sólo por decrementar D de esta manera. Para solucionar esto se podría tratar de incorporar un mecanismo que controle cómo evolucionar el tamaño de las hiperesferas para mantener un decremento lineal más preciso. Una rama de investigación más tiene que ver con la escalabilidad. Con los experimentos realizados se pudo observar que conforme la k aumenta, el número de generaciones evolucionadas se reduce drásticamente. Presentar un algoritmo escalable tanto en las k como en el número de nodos es deseable y esto se puede abordar trabajando en las estructuras de datos usadas o cambiando algunas de las componentes más costosas. Finalmente, se ve razonable pensar que se puede paralelizar la propuesta para reducir el tiempo necesario para obtener soluciones de alta calidad como las encontradas y esto podría además ayudar a mejorar la escalabilidad.

Referencias

[1] http://chriswalshaw.co.uk/partition/ Last visited 03/03/17, 2000.

[2] David A. Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner, editors. *10th DIMACS Implementation Challenge Workshop on Graph Partitioning and Graph Clustering Proceedings*, volume 588 of *Contemporary Mathematics*, Atlanta, GA, USA, 2013. American Mathematical Society.

[3] Charles-Edmond Bichot. *Population-Based Metaheuristics, Fusion-Fission and Graph Partitioning Optimization*, pages 163–199. John Wiley & Sons, Inc., 2013.

[4] Thang Nguyen Bui and Byung Ro Moon. Genetic algorithm and graph partitioning. *IEEE Transactions on Computers*, 45 (7):841–855, July 1996. ISSN 0018-9340.

[5] P. Chardaire, M. Barake, and G. P. McKeown. A probe-based heuristic for graph partitioning. *IEEE Transactions on Computers*, 56(12):1707–1720, Dec 2007.

[6] A. E. Eiben and J. E. Smith. *Hybridisation with Other Techniques: Memetic Algorithms*, pages 173–188. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[7] Philippe Galinier and Jin-Kao Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999. ISSN 1573-2886.

[8] Philippe Galinier, Zied Boujbel, and Michael CoutinhoFernandes. An efficient memetic algorithm for the graph partitioning problem. *Annals of Operations Research*, 191(1):1–22, 2011.

[9] Philippe Galinier, Zied Boujbel, and Michael Coutinho Fernandes. An efficient memetic algorithm for the graph partitioning problem. *Annals of Operations Research*, 191(1):1–22, Nov 2011. ISSN 1572-9338.

[10] Georges R. Harik. Finding multimodal solutions using restricted tournament selection. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 24–31, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. ISBN 1-55860-370-0.

[11] Matthias Hein and Simon Setzer. Beyond spectral clustering - tight relaxations of balanced graph cuts. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pages 2366–2374, USA, 2011. Curran Associates Inc. ISBN 978-1-61839-599-3.

[12] A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4):345–351, Dec 1987. ISSN 1436-5057.

[13] Minh Nghia Le, Yew-Soon Ong, Yaochu Jin, and Bernhard Sendhoff. Lamarckian memetic algorithms: local optimum and connectivity structure analysis. *Memetic Computing*, 1(3):175, Sep 2009. ISSN 1865-9292.

[14] C. León, G. Miranda, and C. Segura. METCO: A Parallel Plugin-Based Framework for Multi-Objective Optimization. *International Journal on Artificial Intelligence Tools*, 18(4):569–588, 2009.

[15] Fernando G. Lobo, Cláudio F. Lima, and Zbigniew Michalewicz, editors. *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*. Springer, 2007.

[16] Manuel Lozano, Francisco Herrera, and José Ramón Cano. Replacement strategies to preserve useful diversity in steady-state genetic algorithms. *Inf. Sci.*, 178(23):4421–4433, December 2008. ISSN 0020-0255.

[17] H. Maini, K. Mehrotra, C. Mohan, and S. Ranka. Genetic algorithms for graph partitioning and incremental graph partitioning. In *Proceedings of Supercomputing '94*, pages 449–457, Nov 1994.

[18] Ole J. Mengshoel and David E. Goldberg. The crowding approach to niching in genetic algorithms. *Evol. Comput.*, 16(3): 315–354, September 2008.

[19] Peter Merz and Bernd Freisleben. Memetic algorithms and the fitness landscape of the graph bi-partitioning problem. In *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature - PPSN*, pages 765–774. Springer, 1998.

[20] Hari Mohan Pandey, Ankit Chaudhary, and Deepti Mehrotra. A comparative review of approaches to prevent premature convergence in GA. *Applied Soft Computing*, 24:1047 – 1077, 2014.

[21] Peter Sanders and Christian Schulz. *Think Locally, Act Globally: Highly Balanced Graph Partitioning*, pages 164–175. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[22] C. Segura, C. A. Coello Coello, E. Segredo, and A. H. Aguirre. A novel diversity-based replacement strategy for evolutionary algorithms. *IEEE Transactions on Cybernetics*, 46(12):3233–3246, Dec 2016.

[23] C. Segura, S. I. V. Peña, S. B. Rionda, and A. H. Aguirre. The importance of diversity in the application of evolutionary algorithms to the sudoku problem. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 919–926, July 2016.

[24] C. Segura, A. Hernandez, F. Luna, and E. Alba. Improving diversity in evolutionary algorithms: New best solutions for frequency assignment. *IEEE Transactions on Evolutionary Computation*, In Press(99), 2017.

[25] Carlos Segura, Arturo Hernández Aguirre, Sergio Ivvan Valdez Peña, and Salvador Botello Rionda. *The Importance of Proper Diversity Management in Evolutionary Algorithms for Combinatorial Optimization*, pages 121–148. Springer International Publishing, 2017.

[26] Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. Exploration and exploitation in evolutionary algorithms: A survey. *ACM Computing Surveys*, 45(3):35:1–35:33, July 2013.

[27] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, and Christian Prins. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & Operations Research*, 40(1): 475 – 489, 2013.