Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Laboratorio de Tecnologías de Información,
CINVESTAV-Tamaulipas

Metaheurísticas para la Minimización de la Suma del Ancho de Banda Cíclico en Grafos

Tesis que presenta:

María Valentina Narváez Terán

Para obtener el grado de:

Maestro en Ciencias en Computación

Director de la Tesis:
Dr. Eduardo A. Rodríguez Tello

Cd. Victoria, Tamaulipas, México. Agosto, 2016

La tesis presentada por María Valentina Narváez Terán fue aprobada por:

Dr. Ricardo Landa Becerra

Dr. José Gabriel Ramírez Torres

Dr. Eduardo A. Rodríguez Tello, Director

Cd. Victoria, Tamaulipas, México., 09 de Agosto de 2016

Agradecimientos

Agradezco al CINVESTAV-Tamaulipas, a su excelente cuerpo de investigadores, en particular al Dr. Eduardo Arturo Rodríguez Tello, por su guía académica y apoyo constante.

De igual manera agradezco al CONACyT por proveerme soporte económico durante el curso de mis estudios de posgrado.

Esta tesis ha sido financiada por la beca de estudios de maestría CONACyT No. 290915 y la Beca Mixta para estancias en el extranjero No. 291062.

Índice General

Índice de Figuras

Índice de Tablas

Índice de Algoritmos

Resumen

Metaheurísticas para la Minimización de la Suma del Ancho de Banda Cíclico en Grafos

por

María Valentina Narváez Terán
Laboratorio de Tecnologías de Información, CINVESTAV-Tamaulipas
Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, 2016
Dr. Eduardo A. Rodríguez Tello, Director

El Problema de Minimización de la Suma del Ancho de Banda Cíclico (CBSP) en grafos consiste en encontrar un etiquetado que minimice la suma de las diferencias cíclicas entre etiquetas de vértices adyacentes. Actualmente existen pocos algoritmos propuestos para resolver este importante problema. Los resultados experimentales de mach, el mejor algoritmo reportado en la literatura, se limitan a instancias con $n \leq 199$ vértices. En este trabajo de tesis se presentan dos algoritmos para el CBSP: un Algoritmo Memético (MA) y una Búsqueda Básica por Vecindario Variable (BVNS), que permiten resolver eficientemente instancias de hasta $n \leq 5,300$ vértices. El desempeño de dichos algoritmos fue evaluado mediante una extensa experimentación desarrollada con 412 instancias agrupadas en 4 diferentes conjuntos. Los resultados muestran que MA y BVNS son alternativas altamente competitivas para resolver el CBSP, ya que permitieron encontrar un gran número de nuevas cotas superiores para este problema.

Palabras clave: Problema de la Suma del Ancho de Banda Cíclico (CBSP), Algoritmo Memético, Búsqueda Básica por Vecindario Variable.

Abstract

Metaheuristics for Cyclic Bandwidth Sum Minimization on Graphs

by

María Valentina Narváez Terán
Information Technology Laboratory, CINVESTAV-Tamaulipas
Research Center for Advanced Study from the National Polytechnic Institute, 2016
Dr. Eduardo A. Rodríguez Tello, Advisor

The Cyclic Bandwidth Sum Minimization Problem on graphs consists in finding a labeling through which the sum of cyclic diferences between adjacent vertices labels become minimal. At the present day, few proposed algorthims exist in order to resolve this important problem. Experimental results of mach, the best algorithm reported in the literature, are limited to instances of $n \leq 199$ nodes. In this tesis work two algorithms are proposed: Memetic Algorith (AM) and Basic Variable Neighbourhood Search (BVNS), that allow to eficiently solve the CBSP for instances of $n \leq 5,300$ nodes. The performance of both algorithms was evaluated by an extensive experimentation over 412 instances grouped in to 4 sets. The results show that MA and BVNS are competitive alternatives to solve the CBSP, since they are able to improve most of the best-known upper bounds for this problem.

Key words: CBSP, graph labeling, Memetic Algorithms, Basic Variable Neighbourhood Search.

Nomenclatura

CBS	Suma del Ancho de Banda Cíclico
CBSP	Problema de la Suma del Ancho de Banda Cíclico
BSP	Problema de la Suma del Ancho de Banda
BVNS	Búsqueda Básica por Vecindario Variable
MA	Algoritmo Memético
RMSE	Error Cuadrático Medio

1

Introducción

El contenido de este capítulo comprende la definición de los Problemas de Etiquetado de Grafos (GLP, *Graph Labeling Problems*)[1] y sus antecedentes históricos, haciendo hincapié en el principal objeto de estudio de esta investigación: el Problema de la Suma del Ancho de Banda Cíclico (CBSP, *Cyclic Bandwidth Sum Problem*). Se describen los motivos y aplicaciones por los cuales el CBSP es considerado un problema relevante tanto en el área de la teoría de grafos como en la optimización combinatoria. Así mismo, se introduce la hipótesis respecto al uso de algoritmos metaheurísticos para construir soluciones del CBSP cercanas al óptimo y se detallan los objetivos propuestos que llevarán a verificar dicha hipótesis. Finalmente, son presentadas las contribuciones al estado del arte producidas durante el transcurso de esta investigación.

[1] Los acrónimos a lo largo de este documento corresponden a las versiones originales en inglés.

1.1 Antecedentes y motivación

El CBSP pertenece a una categoría de problemas conocidos como Problemas de Etiquetado de Grafos (GLP), cuyas principales características son: a) la asignación de etiquetas a los nodos de un grafo, las cuales definen el embebido de dichos nodos dentro de una estructura específica, e.g., un grafo de tipo camino; y b) el costo de implementar dicho embebido es calculado en términos de la distancia respecto a la estructura en la cual se embeben los nodos.

De acuerdo con la generalización propuesta por Chung en 1988 [7], un GLP puede ser definido en términos de dos grafos: un huésped y un anfitrión. En un GLP se busca etiquetar los vértices de un grafo huésped G utilizando para ello los vértices de un grafo anfitrión H. Este etiquetado define el embebido de los vértices de G en H, el cual está sujeto a alguna de las siguientes condiciones:

1. Minimizar la máxima distancia en el grafo anfitrión entre los vértices adyacentes del huésped.

2. Minimizar la suma total de las distancias entre los vértices adyacentes del huésped.

Los problemas definidos por las condiciones 1 y 2 en los que el grafo anfitrión es un camino son conocidos como el Problema del Ancho de Banda (BP, *Bandwidth Problem*) [20, 37, 44] y el Problema de la Suma del Ancho de Banda (BSP, *Bandwidth Sum Problem*) [1, 14, 20, 31], respectivamente. La versión máx-mín del BP es denominada como el Problema del Anti Ancho de Banda (ABP, *AntiBandwidth Problem*) [33, 57].

El Problema de la Suma del Ancho de Banda Cíclico (CBSP, *Cyclic Bandwidth Sum Problem*), introducido por Jinjiang [29], es una variante del BSP en la cual el grafo anfitrión es un ciclo.

En contraste con el BP y el BSP, el CBSP ha sido poco estudiado. Jianxiu [28] así como Chen y Yan [6] han calculado el valor teórico del óptimo para ciertos tipos de grafos y establecido cotas inferiores y superiores. Satsangi *et al.* [52] propusieron un algoritmo de Búsqueda General por Vecindario Variable (GVNS, *General Variable Neighbourhood Search*) que demostró un buen desempeño en grafos con óptimos conocidos y Hamon *et al.* [18] desarrollaron una heurística para

resolver el CBSP, siguiendo la estructura del grafo, capaz de alcanzar los óptimos teóricos para algunos tipos especiales de grafos.

El estudio del problema CBSP es intrínsecamente relevante en teoría de grafos debido a que pertenece a la clase *NP-Difícil* [36, 58], y hasta donde se ha investigado, no se conocen soluciones exactas para algunos tipos particulares de grafos y tampoco se cuenta con algoritmos aproximados capaces de resolverlo eficientemente en el caso de grafos generales de gran tamaño ($n \geq 450$). Los algoritmos existentes únicamente han permitido calcular límites teóricos para clases especiales de grafos[2]. Otro aspecto teórico que motiva fuertemente este trabajo de investigación es el hecho de que los conocimientos generados a partir del estudio del problema CBSP, en cuanto a las propiedades de su espacio de búsqueda, podrían también ser empleados para la resolución de otros problemas de etiquetado de grafos.

Desde la perspectiva práctica el problema CBSP es importante debido a que existen diversas situaciones reales que pueden ser modeladas como instancias de este problema para su resolución computacional. En particular, en el diseño de circuitos VLSI el CBSP proporciona una cota para la longitud total de los cables cuando un circuito (modelado por un grafo G) es embebido en un microchip (modelado por otro grafo tipo ciclo H) [3, 54].

La Figura 1.1 muestra el embebido de un circuito VLSI representado por el grafo H en un microchip con una arquitectura cíclica modelada por H. La longitud total de las pistas de conexión requeridas equivaldrá a la suma de distancias entre vértices adyacentes de G. Según este ejemplo, 33 unidades de material conector son necesarias; sin embargo esta cantidad podría ser reducida mediante la resolución del CBSP.

Otros ejemplos son el diseño de códigos correctores de errores [20], la simulación de la arquitectura paralela de una computadora mediante otra distinta [41] y la calendarización en redes basadas en multidifusión [35].

El espacio de soluciones de un GLP está dado por los todos los posibles mapeos biyectivos entre

[2] Caminos, ciclos, estrellas, grafos completos, etc.

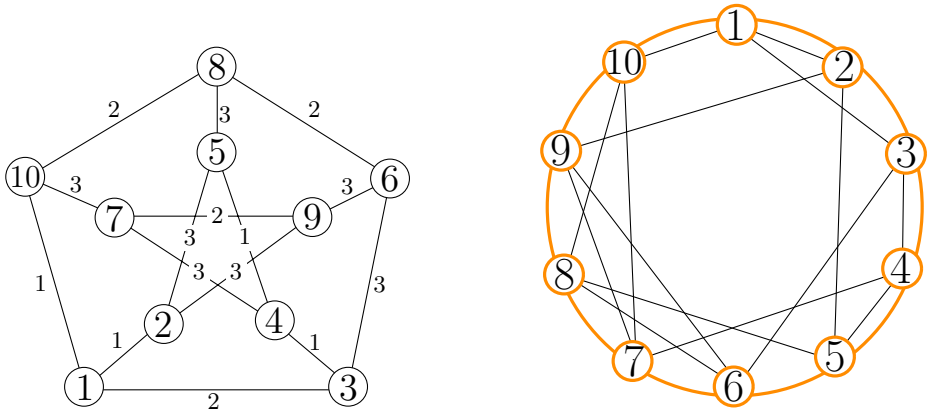


Figura 1.1: a. Circuito VLSI representado mediante el grafo G. b. Circuito embebido en un microchip modelado por H.

los vértices huéspedes y anfitriones. Si un mapeo biyectivo es representado como un vector indexado S en el cual el valor del i-ésimo elemento S_i denota el nodo anfitrión que hospeda al i-ésimo nodo huésped, se tiene que S resulta ser una permutación de n nodos huéspedes. De este modo, el espacio de soluciones equivale a todas las permutaciones de tamaño n, es decir, $n!$ posibles soluciones. Tomando en cuenta la condición cíclica del CBSP el tamaño del espacio de soluciones se reduce a $\frac{(n-1)!}{2}$. El orden factorial del número de posibles soluciones hace impracticable un enfoque de solución exhaustivo, salvo en grafos muy pequeños, por lo que la aplicación de una técnica metaheurística es el método más viable.

Pese a la relevancia de las aplicaciones del problema y la impracticabilidad del enfoque de solución exhaustivo, son escasas las propuestas algorítmicas que permitan resolver el CBSP en el caso general[3]. Por tanto, se encuentra un área de oportunidad importante en la creación de nuevas metaheurísticas que proporcionen soluciones de buena calidad, en un tiempo razonable, para el CBSP en su caso general.

[3]Cualquier tipo de instancia del CBSP, incluyendo grafos que no poseen una estructura definida y para los cuales se desconoce el valor de la solución óptima.

1.2 Hipótesis

En esta investigación se busca verificar la siguiente hipótesis respecto al empleo de un Algoritmo Mémetico y una Búsqueda Básica por Vecindario Variable para resolver el CBSP:

> Los algoritmos metaheurísticos para el CBSP basados en Algoritmos Meméticos y Búsqueda Básica por Vecindario Variable son capaces de proporcionar soluciones que mejoren los resultados de las propuestas actuales, en términos de costo de solución y/o tiempo computacional.

1.3 Objetivos

En esta sección se presenta el objetivo general que fue planteado para comprobar la hipótesis de esta investigación. Así mismo, se detallan los objetivos específicos que permitirán dar cumplimiento al mencionado objetivo general.

1.3.1 Objetivo general

El objetivo general de esta investigación consiste en proponer dos metaheurísticas, una monosolución y una poblacional, basadas en Búsqueda Básica por Vecindario Variable (BVNS) y en Algoritmo Memético (MA), respectivamente, que sean capaces de resolver el caso general del CBSP en instancias con $n \leq 5,300$ vértices y de competir con las propuestas de solución existentes en términos de calidad de solución y eficiencia temporal.

1.3.2 Objetivos específicos

Para alcanzar el objetivo general anteriormente mencionado, se plantean los siguientes objetivos específicos:

- Construir un directorio de instancias del problema que incluya tanto aquellas reportadas anteriormente en la literatura como las generadas en el transcurso de esta investigación, así como recopilar los resultados de las propuestas existentes para dicho grupo de instancias. En especial la mejor solución encontrada, el costo promedio, así como el tiempo promedio de ejecución.

- Implementar una representación de soluciones para MA y BVNS que permita la evaluación directa de estas sin requerir transformaciones adicionales.

- Desarrollar operadores heurísticos inspirados en búsqueda local que durante la fase de mutación del algoritmo MA permitan mejorar la aptitud de los individuos.

- Implementar operadores de selección y supervivencia que equilibren la presión de selección del algoritmo MA.

- Reducir el tiempo de cómputo requerido por la evaluación de las soluciones dotando a los algoritmos MA y BVNS de una función de evaluación incremental.

- Implementar vecindarios que permitan agilizar la generación de soluciones vecinas. Diseñar al menos un vecindario inteligente que explote información inherente al problema.

- Implementar un fase de perturbación basada en cambios incrementales a la solución base.

- Adquirir información sobre el desempeño de las mejores heurísticas para CBSP reportadas en el estado del arte.

- Evaluar y contrastar el desempeño de los algoritmos MA y BVNS contra lo reportado en la literatura, en términos de costo de solución y tiempo de ejecución.

1.4 Contribuciones

Durante el curso de la investigación aquí reportada se obtuvieron dos algoritmos metaheurísticos, un MA y un BVNS, capaces de resolver eficientemente instancias de $n \leq 5,300$ vértices y de producir mejores resultados que los reportados en el estado del arte, respecto al mejor costo de solución, costo promedio de solución y tiempo promedio de ejecución. Además, se conformó un *benchmark* compuesto por 412 instancias para el cual de han registrado los resultados de los algoritmos MA, BVNS y de la heurística MACH reportada en la literatura. Dicho *benchmark* puede ser consultado en línea en la dirección: `http://www.tamps.cinvestav.mx/~ertello/cbsp.php`.

1.5 Organización de la tesis

El resto de esta tesis está conformado por los siguientes cuatro capítulos: El Capítulo 2 incluye un resumen de lo reportado hasta el momento en la literatura acerca del CBSP, incluyendo las cotas teóricas y los diferentes enfoques de solución propuestos. El Capítulo 3 contiene la descripción de las dos alternativas de solución del CBSP propuestas en esta investigación, detallando sus principales componentes. La metodología experimental para evaluar los algoritmos MA y BVNS propuestos, los resultados de dicha experimentación y el análisis de éstos son presentados en el Capítulo 4. Finalmente, el Capítulo 5 contiene las conclusiones de la investigación y las posibles áreas de oportunidad derivadas de las mismas.

2

Estado del arte

En este capítulo se presenta un resumen del conocimiento existente hasta el momento acerca del CBSP, partiendo de su definición formal y abordando el problema desde dos perspectivas principales. Se inicia con el enfoque matemático para el cálculo, ya sea exacto o aproximado, del valor del costo óptimo para grafos con o sin una estructura definida, y se muestra la relación de magnitud entre el CBSP y el BSP. Posteriormente se describen y analizan las dos principales propuestas de solución metaheurística reportadas en la literatura para resolver el CBSP: un algoritmo de búsqueda local GVNS y un algoritmo constructivo denominado MACH.

2.1 Introducción

Los problemas de etiquetado de grafos (GLP, Graph Labeling Problems) pertenecen a la clase de los problemas combinatorios. Desde la década de 1980, los GLP han sido relevantes en el área del diseño de circuitos VLSI [3, 54]. En 1988, Chung [7] utilizó las características comunes de los GLP en la definición de un modelo general para describirlos. Este modelo plantea que todo problema de

etiquetado de grafos consiste en embeber los vértices de un grafo huésped en un grafo anfitrión, de modo que se optimice la distancia entre vértices adyacentes, o la sumatoria de dichas distancias.

El CBSP es un caso de GLP, en el que el grafo anfitrión es un ciclo y tiene como objetivo la minimización de la suma de distancias entre etiquetas de vértices adyacentes. El CBSP fue originalmente propuesto por Jinjiang en 1995 [29]. Yuan [58] demostró que, al igual que el CBP [36], el CBSP pertenece a la clase *NP-Difícil*.

2.2 Definición formal del problema

Formalmente, es posible definir el CBSP como:

Sean el grafo huésped $G(V, E)$ un grafo finito no-dirigido y el anfitrión $H(V', E')$ un grafo tipo ciclo, ambos de orden n. Dada una función biyectiva $\varphi : V \to V'$ que representa un etiquetado de G en H, la Suma del Ancho de Banda Cíclico (CBS) con respecto a φ se define como:

$$CBS(G, \varphi) = \sum_{(u,v)\in E} min(|\varphi(u) - \varphi(v)|, n - |\varphi(u) - \varphi(v)|)\,, \tag{2.1}$$

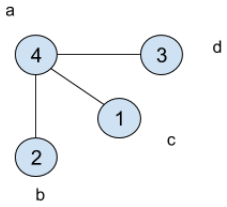
donde $min(|\varphi(u) - \varphi(v)|, n - |\varphi(u) - \varphi(v)|)$ es llamada *distancia cíclica*, y $\varphi(u)$ denota la etiqueta asociada al vértice u. El CBSP consiste en encontrar un etiquetado φ^* tal que $CBS(G, \varphi^*)$ sea mínimo, i.e.,

$$CBS(G, \varphi^*) = min_{\varphi\in\xi}\{CBS(G, \varphi)\}\,, \tag{2.2}$$

donde ξ es el conjunto de todos los posibles etiquetados. El etiquetado que satisface esta condición es denominado etiquetado óptimo.

En la Figura 2.1.a se muestra un grafo G a cuyos nodos $V = \{a, b, c, d\}$ les han sido asignadas etiquetas de acuerdo con la función biyectiva $\varphi : V \to V'$ definida en la Figura 2.1.b. El embebido de G en H definido por φ es representado en la Figura 2.1.c).

$G = (V, E)$



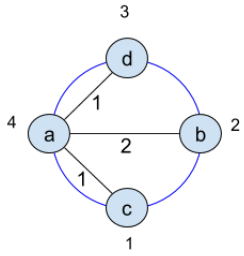
a)

$\varphi : V \to V'$

$u \in V$	$\varphi(u) \in V'$
a	4
b	2
c	1
d	3

b)

$H = (V', E')$



c)

Figura 2.1: a) Grafo huésped $G = (V, E)$ con etiquetas al interior de los nodos. b) Etiquetado φ relacionado los nodos huéspedes $V = \{a, b, c, d\}$ con sus respectivos nodos anfitriones $V' = \{1, 2, 3, 4\}$. c) Grafo tipo ciclo $H = (V', E')$ hospedando al grafo G según el etiquetado φ. Los pesos de las aristas E representan las distancias cíclicas cuya sumatoria equivale al costo, para este ejemplo $CBS(G, \varphi) = 4$.

2.3 Espacio de búsqueda

El espacio de búsqueda ξ del CBSP está dado por todos los posibles etiquetados φ. Es posible plantear el cálculo del número de posibles etiquetados como un problema de conteo consistente en distribuir n objetos en un estructura circular con n localidades, asignando un único objeto a una única localidad. Las soluciones de este tipo de problema de conteo constituyen permutaciones circulares, y el número de ellas puede ser calculado en función del número de objetos n como:

$$|\xi| = \frac{(n-1)!}{2}. \tag{2.3}$$

2.4 Cotas conocidas

En el estudio del CBSP algunos de los primeros esfuerzos se dirigieron a la naturaleza matemática del problema, particularmente a calcular el valor, ya sea exacto o aproximado, del óptimo según el tipo de grafo. Existen casos en los que el valor del óptimo no puede ser calculado de manera exacta. Para este tipo de grafos se definen cotas inferiores y superiores que delimitan el costo de la solución óptima dentro de un rango de valores.

En cuanto al cálculo del valor del óptimo, existen tres divisiones principales, según la precisión y el tipo de grafo: cotas para grafos de cualquier tipo, cotas para grafos con una cierta estructura y valor exacto para grafos con una estructura específica.

2.4.1 Cotas para grafos generales

En 2001, Jianxiu [28] determinó la cota superior para cualquier tipo de grafo, calculada en función del número de vértices n y el número de aristas e:

$$CBS(G) \leq \frac{e\left\lfloor \frac{n}{2} \right\rfloor \left\lceil \frac{n}{2} \right\rceil}{n-1}\,. \tag{2.4}$$

2.4.1.1. Relación entre BSP y CBSP

En 2007, Chen y Yan [6] relacionaron el BSP[1] y el CBSP para cualquier clase de grafo, como se muestra en la Ecuación 2.5.

$$\frac{BS(G)}{2} \leq CBS(G) \leq BS(G)\,, \tag{2.5}$$

donde $BS(G, \varphi)$ es la Suma del Ancho de Banda del grafo G para un etiquetado particular φ y puede ser calculada como:

[1]Problema de la Suma del Ancho de Banda, Bandwidth Sum Problem.

$$BS(G,\varphi) = \sum_{(u,v)\in E} |\varphi(u) - \varphi(v)| \,. \tag{2.6}$$

2.4.2 Cotas para grafos con estructura

Jianxiu [28] estableció las cotas para grafos generados mediante el producto cartesiano de dos grafos. Si se denota como $V(G_n)$ y $E(G_n)$ al conjunto de vértices y aristas de un grafo G_n de grado n, entonces el producto cartesiano de dos grafos G_n y H_m, denotado $G_n \times H_m$ es el grafo con el conjunto de vértices $V(G_n) \times V(H_m)$ donde $[u_1, v_1]$ es adyacente a $[u_2, v_2]$ si $(u_1, u_2) \in E(G_n)$ y $v_1 = v_2$ o $(v_1, v_2) \in E(H_m)$ y $u_1 = u_2$. A continuación se muestran las cotas para los grafos resultado del producto cartesiano de dos grafos, de grados n y m, de tipo camino, ciclo y grafo completo.

Producto cartesiano de caminos $P_m \times P_n$:

$$CBS(P_m \times P_n) \leq m(n-1) + n^2(m-1), m \geq n \,. \tag{2.7}$$

Producto cartesiano de ciclos $C_m \times C_n$:

$$CBS(C_m \times C_n) \leq m(n^2 - 2n - 2), m \geq n \geq 3 \,. \tag{2.8}$$

Producto cartesiano de grafos completos $K_m \times K_n$:

$$CBS(K_m \times K_n) \leq \frac{1}{6}mn\Big(n^2 + 3n\Big\lfloor\frac{m}{2}\Big\rfloor\Big\lceil\frac{m}{2}\Big\rceil - 1\Big), m \geq n \,. \tag{2.9}$$

Producto cartesiano de un camino y un grafo completo $P_m \times K_n$:

$$CBS(P_m \times K_n) \leq \frac{1}{2}m^2n\Big\lfloor\frac{n}{2}\Big\rfloor\Big\lceil\frac{n}{2}\Big\rceil + n(m-1) \,. \tag{2.10}$$

Producto cartesiano de un camino y un ciclo $P_m \times C_n$:

$$CBS(P_m \times C_n) \leq n(m^2 - m - 1)\,. \tag{2.11}$$

Producto cartesiano de un ciclo y un grafo completo $C_m \times K_n$:

$$CBS(C_m \times K_n) \leq n\Big(\frac{1}{2}m^2\Big\lfloor\frac{n}{2}\Big\rfloor\Big\lceil\frac{n}{2}\Big\rceil + 2m - 2\Big)\,. \tag{2.12}$$

2.4.3 Fórmulas exactas para grafos con estructura

Chen y Yan [6] también establecieron el valor exacto del óptimo para los siguientes tipos específicos de grafos.

Camino: Un grafo compuesto por $n = |V|$ vértices $V = \{v_1, v_2, v_3, ..., v_{n-1}, v_n\}$ que se encuentran conectados por $e = n - 1$ en la forma $E = \{(v_1, v_2), (v_2, v_3), (v_{n-1}, v_n)\}$.

$$CBS(P_n) = n - 1\,. \tag{2.13}$$

Ciclo: Un grafo compuesto por n vértices $V = \{v_1, v_2, v_3, ..., v_{n-1}, v_n\}$ conectados por $e = n$ aristas en la forma $E = \{(v_1, v_2), (v_2, v_3), (v_{n-1}, v_n), (v_n, v_1)\}$.

$$CBS(C_n) = n\,. \tag{2.14}$$

Rueda: Un grafo W_n compuesto por un ciclo C_{n-1} y un vértice $v_n \in V$, tal que $(v_i, v_n) \forall v_i \in C_{n-1}, v_n \notin C_{n-1}$. Posee n vértices y $e = 2n - 2$ aristas.

$$CBS(W_n) = n + \Big\lfloor\frac{1}{4}n^2\Big\rfloor\,. \tag{2.15}$$

Completo: Un grafo K_n con n vértices en el que existe una arista entre cada par de vértices.

$$CBS(K_n) = \begin{cases} \frac{n^3}{8} & n \text{ es par} \\ \frac{n(n-1)(n+1)}{8} & n \text{ es impar} \end{cases}. \tag{2.16}$$

Bipartita Completo: Un grafo $K_{x,y}$ compuesto por dos conjuntos disjuntos de vértices X y Y con cardinalidad $|X| = x$ y $|Y| = y$. Entre cada par de vértices u y v existe una arista si y sólo si u y v pertenecen a conjuntos distintos.

$$CBS(K_{x,y}) = \begin{cases} \frac{xy^2+x^2y}{4} & x, y \text{ pares} \\ \frac{xy^2+x^2y+x}{4} & x \text{ par, } y \text{ impar} \\ \frac{xy^2+x^2y+x+y}{4} & x, y \text{ impares} \\ \frac{xy^2+x^2y+x}{4} & x \text{ impar, } y \text{ par} \end{cases}. \tag{2.17}$$

Potencias de ciclos: Sea el grafo tipo ciclo C_n de grado n. La k-esíma potencia de C_n, denotada C_n^k es un grafo de grado n, en el cual dos vértices son adyacentes si la distancia, calculada como la longitud del camino más corto entre ellos, es máximo de tamaño k.

$$CBS(C_n^k) = \frac{1}{2}nk(k+1), 1 \leq k \leq \left\lfloor \frac{n-1}{2} \right\rfloor. \tag{2.18}$$

2.5 Algoritmos reportados en la literatura

Existen pocas propuestas algorítmicas para resolver el CBSP en el caso general. En 2013 Satsangi [51] presentó un enfoque metaheurístico para resolver el CBSP en grafos estándar con óptimos conocidos y grafos aleatorios. Esta propuesta aplica un conjunto de técnicas que incluye Búsqueda Dispersa Modificada (MSS), Búsqueda General por Vecindario Variable (GVNS), una

variante de GVNS, Búsqueda de Vecindario Variable Reducido (RVNS) y un Algoritmo Genético (GA). Satsangi [51] propone un abanico de heurísticas, operadores de recombinación y funciones de vecindario para las diferentes técnicas de búsqueda. En términos generales, los mejores resultados fueron obtenidos mediante una combinación de GVNS y RVNS presentada con mayor detalle por Satsangi *et al.* en [52].

Los resultados obtenidos por Satsangi *et al.* en [52] fueron superados, en la mayoría de los casos, en 2014 y 2016 por Hamon *et al.* [18, 19], utilizando una heurística constructiva basada en el particionamiento de la estructura del grafo, a la que sus autores denominaron MACH.

Los aspectos más relevantes de ambas propuestas son descritos a detalle y discutidas a continuación.

2.5.1 GVNS

Satsangi *et al.* propusieron en [52] un algoritmo GVNS que además incorpora RVNS como método de mejoramiento de la solución inicial, previo a la aplicación de GVNS. La solución inicial consiste en un etiquetado canónico, es decir, el primer vértice recibe la etiqueta 1, el segundo la etiqueta 2 y así sucesivamente, hasta el n-ésimo vértice que recibe la etiqueta n.

RVNS mejora la solución inicial a través de un número max_{RVNS} de iteraciones utilizando un conjunto de seis operadores de vecindario $NO_k, k = 1, ..., k_{max}$, $k_{max} = 6$. Cada iteración inicia con la aplicación del vecindario $NO_k, k = 1$ para obtener una nueva solución que sustituye a la actual si su costo es menor. En caso contrario, el algoritmo continua aplicando el siguiente vecindario NO_{k+1}. La próxima iteración inicia después de que los seis vecindarios han sido aplicados.

La mejor solución resultante de RVNS es mejorada mediante el proceso iterativo de GVNS. Dicho proceso inicia con una fase de perturbación en la que se obtiene una nueva solución aplicando el operador SPO_k [2].

A continuación, en la fase de búsqueda local, dicha solución es mejorada utilizando sucesivamente

[2]Los operadores de perturbación SPO son los operadores de búsqueda local NO utilizados en la fase de RVNS.

dos operadores de búsqueda $LSO_l, l = 1, ..., l_{max}$, $l_{max} = 2$. Al final de esta fase de búsqueda local, la solución resultante sustituye a la actual si su costo es menor, y se inicia una nueva fase de perturbación, utilizando SPO_k como operador. En caso contrario, la fase de perturbación utilizará SPO_{k+1} como operador de perturbación. Una iteración de $GVNS$ concluye cuando han sido aplicados todos los operadores de perturbación SPO_k y todos los operadores de búsqueda local LSO_l. En total, en GVNS son realizadas max_{GVNS} iteraciones, cada una con seis operadores de perturbación y dos operadores de búsqueda local.

Mediante GVNS, Satsangi *et al.* en [52] alcanzó resultados óptimos para grafos estándar con $n \leq 72$ vértices y estableció el primer *benchmark* para grafos con óptimos desconocidos de $n \leq 199$ vértices. La experimentación de Satsangi *et al.* en [52] puso a prueba la precisión de las cotas inferiores y superiores para grafos generales y productos cartesianos, demostrando que el óptimo puede encontrarse dentro del rango establecido por éstas, pero bastante lejos de los extremos.

En el momento de la publicación del algoritmo GVNS de Satsangi *et al.* [52] no existían resultados reportados para los grafos cuyo valor exacto del óptimo no puede ser calculado, pero como demostraría Hamon*et al.* [19] más adelante, el desempeño de GVNS en este tipo de grafos aún puede ser mejorado significativamente, tanto en costo mínimo de CBS como en tiempo de ejecución.

2.5.2 Heurístca mach

La heurística MACH, propuesta por Hamon et al. [19], opera en dos fases: el particionamiento del grafo en caminos independientes siguiendo un criterio de similitud entre nodos y el etiquetado de dichos caminos de forma que el CBS sea minimizado.

La localización de estos caminos es realizada mediante Búsqueda Primero en Profundidad (BFS) que inicia en el vértice no visitado con menor grado y continua con el nodo vecino no visitado con mayor similitud según el índice de Jaccard [27]. La construcción de un camino concluye cuando el último nodo vistado no posee nodos vecinos sin visitar. Este proceso de particionamiento en caminos es repetido hasta que todos los nodos hayan sido visitados e incluidos en un camino.

La construcción de la solución es realizada incrementalmente mediante la inserción iterativa de caminos. El camino de mayor longitud P_1 es insertado en una lista vacía. A continuación, en orden según su longitud, cada camino es insertado consecutivamente, probando colocar el camino en orden anverso e inverso en cada posible posición de inserción en la lista. La mejor alternativa para insertar cada camino es determinada mediante la evaluación parcial del CBS, eligiendo la que tenga un menor costo.

La propuesta de Hamon *et al.* [19] explota la idea de que el camino es el tipo de grafo con menor CBS, y supone que descomponiendo el grafo en caminos, y minimizando el CBS de éstos, el CBS global será minimizado. Este enfoque mejora los resultados de la propuesta de Satsangi *et al.* [52], sin embargo, aún es probable que existan soluciones de costo menor que las alcanzados mediante la optimización de caminos independientes.

Además, MACH es un método de naturaleza puramente constructiva que deja de lado la posibilidad de mejorar la solución. Si bien esto repercute positivamente en el tiempo de ejecución, impacta negativamente en la calidad de la solución final, especialmente en grafos carentes del nivel de estructura del que MACH depende, e.g., en grafos aleatorios.

2.6 Resumen del capítulo

En este capítulo se han presentado los principales avances en el estudio del CBSP, los referentes al cálculo del valor del óptimo y especialmente las propuestas algorítmicas. El análisis de la literatura de CBSP muestra que existen áreas de oportunidad en la creación de técnicas capaces de resolver instancias de mayor tamaño y/o con poca estructura, para las cuales el óptimo es desconocido, y sugiere que es posible mejorar los resultados reportados hasta el presente.

En el siguiente capítulo de esta tesis se presentan a detalle los principales componentes de las propuestas de solución poblacional y monosolución, basadas en MA y BVNS, que fueron desarrolladas durante el transcurso de esta investigación.

3

Enfoques de solución propuestos

Como parte central de esta investigación se proponen dos técnicas metaheurísticas para la resolución del CBSP. Bajo el enfoque poblacional, fue diseñado e implementado un Algoritmo Mémetico inspirado en la combinación de las técnicas propias de los Algoritmos Genéticos y la Búsqueda Local. Por otra parte, bajo el enfoque monosolución se desarrollo un algoritmo basado en la Búsqueda Básica por Vecindario Variable.

En este capítulo se describen los componentes más relevantes de los algoritmos propuestos: su esquema general, el proceso de inicialización de soluciones, las funciones de evaluación, los operadores evolutivos y de vecindario así como los criterios de paro de cada uno.

3.1 Algoritmo Memético

Los Algoritmos Genéticos, propuestos por Holland en 1975 [21] y difundidos por [16], son una técnica de optimización inspirada en el funcionamiento de la evolución de los organismos biológicos y los principales mecanismos de ésta: selección, cruza, mutación y supervivencia del más apto. La

selección permite que los individuos más aptos de la población cuenten con mayores posibilidades de sobrevivir y heredar sus genes a futuras generaciones mediante la cruza. Por otro lado, a través de la mutación nuevos genes son introducidos aportando así diversidad a la población. Gracias a la acción de estos mecanismos se espera que tras un determinado período temporal la aptitud promedio de los individuos en la población haya mejorado.

La Búsqueda Local se basa en la exploración iterativa del espacio de soluciones a partir de una solución base y una función de vecindario [23, 32]. Mediante la función de vecindario se obtienen nuevas soluciones candidatas, vecinas de la solución base y la más apta de dichas soluciones candidatas es seleccionada como el siguiente paso en la exploración sustituyendo a la solución base [4, 15, 40, 55, 56].

Tanto los Algoritmos Genéticos como la Búsqueda Local han demostrado ser exitosos en la resolución de una amplia variedad de problemas de optimización [9, 32]. Por ello, se propone el uso de un Algoritmo Memético [30, 42, 43], combinando los mecanismos de los Algoritmos Genéticos y la Búsqueda Local para abordar el CBSP.

En el Algoritmo 3.1 se muestra el esquema básico del Algoritmo Memético implementado en esta investigación. Cada uno de sus componentes es descrito con detalle en las siguientes secciones.

Algoritmo 3.1 algoritmoMeméticoCBSP(G, μ, CP, MP, IP)

1: $t \leftarrow 1$
2: $P \leftarrow$ poblaciónInicial(μ)
3: **while** no se cumpla algún criterio de paro **do**
4: $P' \leftarrow$ selección(G, P)
5: $O \leftarrow$ cruza(G, P')
6: $O' \leftarrow$ mutación(G, O)
7: $P \leftarrow$ supervivencia(P, O')
8: $P \leftarrow$ inversión(G, P)
9: $t \leftarrow t + 1$
10: **end while**
11: **return** Individuo más apto $p^* \in P$

3.1.1 Codificación de las soluciones

En el ámbito del Algoritmo Memético, el cromosoma de un individuo es representado como un vector indexado, formado por n elementos o genes. La posición de un gen en el vector, dada por su índice, determina el vértice al cual se asocia la etiqueta dada por el alelo. Es decir, en la población P integrada por μ individuos, según el etiquetado definido por el individuo $p_i \in P$, al vértice $u \in V$ le corresponde la etiqueta $p_i[u]$.

Es posible apreciar que en esta clase de codificación los vectores indexados representan permutaciones de n elementos.

3.1.2 Población inicial

La población inicial es generada de manera pseudoaleatoria mediante el algoritmo Fisher-Yates [13]. Inicialmente son creados μ individuos cuyo cromosoma equivale a la permutación canónica[1]. Posteriormente, los genes de cada individuo $p_i \in P$ son desordenados mediante n intercambios del gen j, $j = 1, ..., n$ con un gen k aleatoriamente seleccionado, como se ilustra en el Algoritmo 3.2.

Algoritmo 3.2 poblaciónInicial(μ)

1: $P \leftarrow \emptyset$
2: **for** $i \leftarrow 1$ **to** μ **do**
3: $p_i \leftarrow \{1, 2, 3, ..., n-1, n\}$
4: **for** $j \leftarrow 1$ **to** n **do**
5: Elegir un gen al azar $k \in p_i$, $k \geq j$
6: $p_i \leftarrow$intercambio(p_i, j, k)
7: **end for**
8: $P \leftarrow P \cup p_i$
9: **end for**
10: **return** P

[1]Permutación de n elementos en el rango $[1, n]$ ordenados ascendentemente.

3.1.3 Función de aptitud

Esta función tiene el propósito de cuantificar la aptitud de los individuos. Dependiendo de la naturaleza del problema y los operadores genéticos empleados, la función de aptitud no siempre equivale directamente a la función objetivo. En este caso, la función de evaluación mostrada en el Algoritmo 3.3 es una codificación directa de la función objetivo de la Ecuación 2.1, en la que las aristas $e_j \in E$ son recorridas una a una y sus distancias cíclicas $cb[e_j]$ son almacenadas y acumuladas. Dado que el CBSP es un problema de minimización, se prefieren los individuos con un menor $CBS(G, p_i)$.

Algoritmo 3.3 CBS(G, p_i)

1: $cbs \leftarrow 0$
2: **for each** $e_j : (u, v) \in E$ **do**
3: $\quad cb[e_j] \leftarrow min(|p_i[u] - p_i[v]|, n - |p_i[u] - p_i[v]|)$
4: $\quad cbs \leftarrow cbs + cb[e_j]$
5: **end for**
6: **return** cbs

Por otro lado, con base en las ideas presentadas en [47–49] fue diseñada una función de evaluación incremental capaz de actualizar la aptitud de un individuo en el que sólo han cambiado los alelos de un subconjunto de genes X, por ejemplo, durante una mutación. Hacer uso de este método permite ahorrar tiempo computacional, al evitar calcular las distancias cíclicas para la totalidad de las aristas. Como se ilustra en el Algoritmo 3.4, sólo son revaluadas las aristas relacionadas con los vértices representados por los genes del conjunto X y actualizadas sus respectivas distancias cíclicas $cb[e_j]$.

Algoritmo 3.4 cbsLocal(G, p_i, X)

1: **for each** $\{e_j : (u, v) \in E | u \in X \oplus v \in X\}$ **do**
2: $\quad cbs \leftarrow cbs - cb[e_j]$
3: $\quad cb[e_j] \leftarrow min(|p_u - p_v|, n - |p_u - p_v|)$
4: $\quad cbs \leftarrow cbs + cb[e_j]$
5: **end for**
6: **return** cbs

3.1.4 Selección

El operador de selección determina cuáles de los individuos en la población P son los más aptos para heredar sus genes a una nueva generación de individuos O. La selección es realizada mediante un torneo binario [17] de 2μ rondas. En cada ronda son elegidos de la población dos individuos $p_j, p_k \in P$ al azar, y el más apto apto de ellos es seleccionado para formar parte de la población de padres P', como se muestra en el Algoritmo 3.5. En este tipo de operador la presión de selección es suavizada, dado que un individuo cuya aptitud sea inferior al promedio aún tiene oportunidad de ser seleccionado, si compite con un individuo de menor aptitud. Por otro lado, es posible asegurar que el individuo con la menor aptitud en la población P nunca será seleccionado, mientras que el mejor será seleccionado siempre que participe en alguna ronda del torneo.

Algoritmo 3.5 selección(G, P)

1: $P' \leftarrow \emptyset$
2: **for** $i \leftarrow 1$ **to** 2μ **do**
3: Elegir aleatoriamente dos individuos p_j, p_k con $1 \leq j, k \leq n$ y $j \neq k$
4: **if** $CBS(G, p_j) < CBS(G, p_k)$ **then**
5: $P' \leftarrow P' \cup p_j$
6: **else**
7: $P' \leftarrow P' \cup p_k$
8: **end if**
9: **end for**
10: **return** P'

3.1.5 Cruza

La recombinación de los genes de los individuos más aptos es uno de los aspectos de mayor importancia dentro de un Algoritmo Genético. A través del mecanismo de cruza, ilustrado en el Algoritmo 3.6, se espera que los genes más benéficos para la aptitud de los individuos se esparzan en la población mediante herencia. Los individuos de la población de padres P' son *emparejados*

según el orden en el que fueron elegidos. Cada pareja de individuos p_i, p_{i+1} tiene una probabilidad CP de producir un descendiente o mediante cruza. En caso de que dicha probabilidad no se cumpla, en lugar de un descendiente producto de cruza, se inserta en la población O un clon del individuo padre, ya sea p_i o p_{i+1}, que posea la mejor apitud.

Algoritmo 3.6 cruza(G, P')

```
1: O ← ∅
2: i ← 1
3: for each p_i, p_{i+1} ∈ P' do
4:    if CP ≥ random(0,1) then
5:       o ← cruzaBasadaEnOrden(p_i, p_{i+1})
6:       O ← O ∪ o
7:    else
8:       if CBS(p_i) < CBS(p_{i+1}) then
9:          O ← O ∪ p_i
10:      else
11:         O ← O ∪ p_{i+1}
12:      end if
13:   end if
14:   i ← i + 2
15: end for
16: return O
```

3.1.5.1. Cruza basada en el orden

La idea subyacente de la cruza basada en el orden [50] es que los individuos hijo producidos mediante ella, hereden bloques de genes en el mismo orden relativo en el que dichos genes se presentan en los individuos padre.

Sean el individuo descendiente o, a y b dos números aleatorios $1 \leq a, b \leq n$ y $[a, b]$ la fracción del cromosoma de un individuo comprendida entre los genes a-esimo y b-esimo. El individuo o hereda los genes indexados por $[a, b]$ directamente del individuo padre p_i, es decir, $o[a, b] \leftarrow p_i[a, b]$. El resto del cromosoma será completado con los genes del individuo p_{i+1} que aún no están presentes en o,

según el orden de aparición en p_{i+1} de dichos genes. El operador de cruza basada en el orden es presentado en el Algoritmo 3.7.

Algoritmo 3.7 CruzaBasadaEnOrden(p_i, p_{i+1})

1: $a \leftarrow random(1, n)$
2: $b \leftarrow random(1, n)$
3: $l \leftarrow 1$
4: $h \leftarrow \emptyset$
5: Marcar genes $o[1, n]$ como indefinidos
6: $o[a, b] \leftarrow p_i[a, b]$
7: $h \leftarrow h \cup p_i[a, b]$
8: Marcar genes $o[a, b]$ como definidos
9: **for each** $k \leftarrow 1$ **to** n **do**
10: **if** $o[k]$ se encuentra indefinido **then**
11: **while** $l \leq n$ **do**
12: **if** $p_{i+1}[l] \notin h$ **then**
13: $o[k] \leftarrow p_{i+1}[l]$
14: $h \leftarrow h \cup p_{i+1}[l]$
15: Marcar gen $o[k]$ como definido
16: **break**
17: **else**
18: $l \leftarrow l + 1$
19: **end if**
20: **end while**
21: **end if**
22: **end for**
23: **return** o

3.1.6 Mutación

Mientras que la cruza de individuos tiene por objeto la proliferación de los genes benéficos prexistentes en la población, la mutación es el mecanismo que permite introducir nuevo material genético. Este nuevo material genético se da en forma de alteraciones a los genes existentes, las

cuales se producen con una probabilidad MP. En el caso de las representaciones binarias, uno de los operadores de mutación más populares es la inversión bit a bit [11]. Sin embargo, en el caso de una representación basada en permutaciones, como la utilizada en esta propuesta, el operador de mutación debe ser adecuado para evitar producir cromosomas inválidos. Por éste motivo se optó por una mutación basada en intercambios sucesivos de dos genes [8]. Según se especifica en el Algoritmo 3.8, al mutar un individuo $o_i \in O$, cada uno de sus genes $o_i[j]$ es intercambiado con un gen aleatorio $o_i[k]$. Tras cada intercambio, la aptitud del individuo es evaluada tomando en cuenta sólo los vértices afectados $X = \{j, k\}$, como se definió en el Algoritmo 3.4. El individuo original o es sustituido por el individuo mutado o' si la aptitud se vio beneficiada por dicho intercambio.

Algoritmo 3.8 mutación(G, O)

```
1: O' = ∅
2: for each o_i ∈ O do
3:    if MP ≥ random(0,1) then
4:       for j ← 1 to n do
5:          k ← random(1,n)
6:          o'_i ← intercambio(o, j, k)
7:          X ← {j, k}
8:          if cbsLocal(G, o'_i, X) <= CBS(G, o) then
9:             o_i ← o'_i
10:         end if
11:      end for
12:   end if
13:   O' ← o_i
14: end for
15: return O'
```

3.1.7 Supervivencia

La fase de la supervivencia determina cuáles de los individuos presentes en la población durante la generación t seguirán presentes en la generación $t+1$. Existen dos enfoques principales: supervivencia

(μ, λ) basada en la aptitud y supervivencia $(\mu + \lambda)$ basada en la edad [11]. Bajo el primer enfoque, los individuos en la población de descendientes O sustituye directamente a la población P, sin tomar en cuenta la aptitud. Por otro lado, en la supervivencia $(\mu + \lambda)$, padres P y descendientes O forman una población conjunta R, de la cual sobreviven sólo los individuos más aptos. El primer enfoque fue elegido para esta propuesta ya que consume menos tiempo computacional y permite evitar que la población se estanque en óptimos locales.

3.1.8 Inversión

El último operador aplicado es el operador de inversión descrito en el Algoritmo 3.9. Este actúa, con probabilidad IP, sobre cada individuo $p_i \in P$ eligiendo dos genes al azar j y k. Partiendo de los extremos hacia el centro, el orden de los genes comprendidos entre tales puntos es invertido iterativamente. Tras cada inversión, se determina si se produjo o no mejora en la aptitud del individuo. En el caso afirmativo, la inversión es conservada, y revertida en caso contrario. Al final de esta fase, el individuo p conservará únicamente las inversiones que fueron benéficas para su aptitud.

Algoritmo 3.9 inversión(G, P)

```
1: for each p_i ∈ P do
2:    if IP ≥ random(0,1) then
3:       Elegir dos números aleatorios j,k|1 ≤ j,k ≤ n
4:       while j! = k do
5:          p'_i ← intercambio(p_i, j, k)
6:          j ← j + 1
7:          k ← k − 1
8:          X ← {j, k}
9:          if cbsLocal(G, p'_i, X) <CBS(G, p_i) then
10:            p_i ← p'_i
11:         end if
12:      end while
13:   end if
14: end for
15: return P
```

3.1.9 Criterios de paro

El Algoritmo Memético termina su ejecución cuando han transcurrido un número t_{max} de generaciones, o bien, cuando se ha alcanzado un tiempo de ejecución máximo predeterminado en segundos.

3.2 Búsqueda Básica por Vecindario Variable

BVNS es un algoritmo de búsqueda local basado en alteraciones de la solución inicial y cambios sistemáticos de las estructuras de vecindario. El algoritmo desarrollado sigue el siguiente esquema: una solución inicial s es proporcionada como entrada a un método iterativo de tres fases: perturbación, mejora mediante búsqueda local y cambio de vecindario.

En esta investigación se propone un algoritmo BVNS cuya solución inicial es generada utilizando uno de tres posibles métodos: aleatoriamente, mediante un Algoritmo Avaro desarrollado en esta investigación o mediante la heurística MACH [19]; la fase de perturbación realiza un número incremental de movimientos de intercambio y la búsqueda local utiliza de forma alternada dos operadores de vecindario NG_i. El esquema general del algoritmo BVNS implementado es descrito en el Algoritmo 3.10 mostrado a continuación.

Algoritmo 3.10 BVNS(G)

```
1: s ← soluciónInicial()
2: t ← 1
3: kM ← 1
4: i ← 1
5: while no se cumpla alguno de los criterios de paro do
6:    s' ← perturbación(s, kM)
7:    s'' ← búsquedaLocal(s', NG_i)
8:    while CBS(G, s'') < CBS(G, s') do
9:       s' ← s''
10:      if CBS(G, s') <CBS(G, s) then
11:         s ← s'
12:         kM ← 1
13:      else
14:         kM ← kM + 1
15:      end if
16:      s'' ← búsquedaLocal(s', NG_i)
17:   end while
18:   i ← tmod2
19:   t ← t + 1
20: end while
21: return s
```

3.2.1 Solución inicial

Para propósitos comparativos, se cuenta con tres alternativas para la generación de la solución inicial: aleatoria, mediante un Algoritmo Avaro y mediante la heurística MACH [19]. En el Capítulo 4 se compara y analiza el impacto del método de inicialización de la solución utilizado en el desempeño del algoritmo BVNS.

3.2.2 Inicialización aleatoria

La inicialización de solución aleatoria sigue un esquema similar al descrito en el Algoritmo 3.2, en el que se parte de una permutación canónica cuyo orden es alterado mediante una serie de intercambios aleatorios sucesivos.

3.2.3 Inicialización mediante la heurística mach

El código fuente del algoritmo MACH ha sido proporcionado por sus autores, por lo que fue posible utilizar dicho método para obtener soluciones iniciales. La mejor solución producto de 31 ejecuciones del algoritmo MACH fue almacenada y utilizada como punto de inicio para el algoritmo BVNS.

3.2.4 Inicialización mediante un Algoritmo Avaro

El Algoritmo Avaro desarrollado en esta investigación se ha basado en la idea de visitar los vértices de un grafo G mediante un recorrido en anchura, como se muestra en los Algoritmos 3.11 y 3.12. Durante dicho recorrido, los vecinos no visitados de un vértice u reciben la etiqueta $minL$ o $maxL$ que minimice su distancia cíclica respecto a u.

Algoritmo 3.11 inicializaciónAvara(G, $vérticeInicial$)

```
1: n ← Número de vértices en el grafo
2: minL ← 2
3: maxL ← n
4: encolar(vérticeInicial)
5: Marcar vérticeInicial como visitado
6: s[vérticeInicial] ← 1
7: while existan vértices sin visitar do
8:     u ← desencolar()
9:     for each (u, v) ∈ E do
10:        if v no ha sido visitado then
11:            vL ← etiquetar(v, minL, maxL)
12:            if vL = minL then
13:                s[v] ← minL
14:                minL ← minL + 1
15:            else
16:                s[v] ← maxL
17:                maxL ← maxL − 1
18:            end if
19:            Marcar v como visitado
20:            encolar(v)
21:        end if
22:    end for
23: end while
```

Algoritmo 3.12 etiquetar($v, minL, maxL$)

```
1: d1 ← min(minL − v, n − (minLabel − v))
2: d2 ← min(maxL − v, n − (maxLabel − v))
3: if d1 ≤ d2 then
4:     return minL
5: else
6:     return maxL
7: end if
```

El uso de este tipo de inicialización permite iniciar la búsqueda desde un punto de partida mejor que el proporcionado por una construcción aleatoria. Hipotéticamente, al partir de una mejor solución inicial, se requerirían menos iteraciones del algoritmo de búsqueda y menor tiempo para alcanzar una solución de un costo dado, por lo que sería posible encontrar una solución final de mejor calidad.

3.2.5 Función de evaluación

Al igual que el Algoritmo Memético descrito en la Sección 3.1, el algoritmo BVNS cuenta con dos funciones de evaluación: una función global y una función de evaluación incremental, similares a las descritas en el Algoritmo 3.3 y el Algoritmo 3.4 de la Sección 3.1.3.

3.2.6 Fase de perturbación

La perturbación de la solución s es aplicada cuando la búsqueda se ha estancado en una solución que no puede ser mejorada a través de la aplicación del vecindario actual. Esta fase del algoritmo BVNS tiene como objetivo guiar la búsqueda fuera de la cuenca de atracción de un óptimo local s, mediante un tamaño de paso incremental kM. Este tamaño de paso determina el grado de la perturbación, es decir, el número de movimientos que serán aplicados a la solución s para obtener la solución perturbada s'. En el marco del algoritmo BVNS diseñado en esta investigación, un movimiento consiste en el intercambio de las etiquetas de dos vértices u, v elegidos al azar. La fase de perturbación es descrita en el Algoritmo 3.13.

Algoritmo 3.13 perturbación(s, kM)

1: $X \leftarrow \emptyset$
2: **for** $i = 1$ **to** kM **do**
3: $u, v \leftarrow$ Elegir dos vértices aleatorios
4: $s' \leftarrow$intercambio(s, u, v)
5: $s \leftarrow s'$
6: $X \leftarrow X \cup u, v$
7: **end for**
8: $cbsLocal(G, s', X)$
9: **return** s'

3.2.7 Vecindarios

Se implementaron dos funciones de vecindad, NG_1 y NG_2, ambas basadas en el intercambio de etiquetas de dos vértices u y v. La diferencia entre estos vecindarios radica en el orden en que son recorridas las soluciones vecinas.

Como se muestra en la Ecuación 3.1, en el vecindario NG_1 los vértices u y v son obtenidos de manera aleatoria, mientras que en el vecindario NG_2 descrito por la Ecuación 3.2 el vértice u posee un alto $CBS(u, s)$ y el vértice v es aleatorio.

$$NG_1(s) = \{s' = intercambio(s, u, v) | \forall u, v \in I_{aleatorio(V)}, u \neq v\} \tag{3.1}$$

$$NG_2(s) = \{s' = intercambio(s, u, v) | \forall u \in I_{ordenado(V)}, v \in I_{aleatorio(V)}\} u \neq v\,, \tag{3.2}$$

donde $I_{aleatorio(V)}$ es el conjunto de vértices listados aleatoriamente e $I_{ordenado(V)}$ es el conjunto de vértices descendentemente ordenados según su contribución $CBS(u, s)$ al $CBS(G, s)$ dado por la Ecuación 3.3.

$$CBS(u,s) = \sum_{(u,v)\in E} min(|s[u] - s[v]|, n - |s[u] - s[v]|)\,. \tag{3.3}$$

Durante la exploración del vecindario, existen dos alternativas principales para elegir cuál de las soluciones vecinas remplazará a la solución actual. Con el enfoque *primera-mejora*, la primera solución s' vecina visitada que mejore a la solución actual es elegida. De este modo es posible evitar realizar una exploración completa del vecindario, sin embargo, las mejoras son de menor magnitud. Por otro lado, utilizando el enfoque *mayor-mejora* se explora la totalidad del vecindario, evaluando todas las soluciones que lo conforman y eligiendo a la mejor de ellas. Aunque la magnitud de la mejora alcanzada entre las búsquedas t y $t+1$ puede ser mayor mediante *mayor-mejora*, la exploración del vecindario completo puede resultar demasiado demandante en cuanto a tiempo de cómputo.

Dado que las conjuntos NG_1 y NG_2 conforman ambos vecindarios de tamaño $\frac{n(n-1)}{2}$ se eligió el enfoque de *primera-mejora* con el objetivo de reducir el tiempo empleado en la exploración. Cabe mencionar que aún utilizando este enfoque la complejidad de esta fase del algoritmo es cuadrática en el peor caso, *i.e.*, cuando sólo existe una solución que mejora y ésta es visitada en último lugar.

3.2.8 Fase de búsqueda local

La búsqueda local, ilustrada en el Algoritmo 3.14 recorre el vecindario definido por el operador NG_i, visitando las soluciones vecinas de la solución actual. La búsqueda termina cuando, tras recorrer todo el vecindario, no se encontró solución vecina alguna que mejore a la actual.

Algoritmo 3.14 búsquedaLocal(s', NG_i)

```
1: if NG_i = NG_1 then
2:     vértices_u, vértices_v ← I_aleatorio
3: else
4:     vértices_u ← I_aleatorio
5:     vértices_v ← I_ordenado
6: end if
7: for i ← 1 to n do
8:     for j ← 1 to n do
9:         s'' ←intercambio(s', vértices_u[i],vértices_v[j])
10:        X ← {vértices_u[i],vértices_v[j]}
11:        if cbsLocal(G, s'', X) < CBS(G, s') then
12:            return s''
13:        end if
14:    end for
15: end for
16: return s'
```

3.2.9 Criterios de paro

Fueron incorporados dos criterios de paro para el algoritmo BVNS: un número prefijado de búsquedas locales totales y un tiempo de ejecución predefinido. El algoritmo termina al alcanzar el límite máximo para cualquiera de estos dos criterios.

3.3 Resumen

En este capítulo fueron descritos los principales componentes de los algoritmos propuestos en esta investigación para resolver el CBSP. Se detallaron los operadores de inicialización, selección, cruza, mutación e inversión del MA, así como las fases de inicialización, perturbación, búsqueda local y cambio de vecindario del algoritmo BVNS. También fueron descritas las funciones de evaluación y criterios de paro utilizados por ambos algoritmos.

La experimentación realizada para evaluar el desempeño de los algoritmos descritos en este capítulo, así como por la heurística MACH [19] es presentada y analizada en el Capítulo 4.

4

Experimentación y resultados

En este capítulo se presentan las métricas de desempeño, grupos de instancias de prueba, parámetros de entrada de los algoritmos, así como las condiciones experimentales empleadas para realizar la comparación de los algoritmos MA y BVNS propuestos en esta investigación, contra el algoritmo MACH, el cual produce los mejores resultados reportados hasta el momento en la literatura.

Los tres algoritmos fueron ejecutados utilizando un total de 412 instancias divididas en cuatro conjuntos: *Productos-Cartesianos*, *Harwell-Boeing*, *Estándar* y *Aleatorios*. Para cada conjunto de instancias se presenta un análisis detallado de sus resultados.

4.1 Criterios de comparación

Con el propósito de medir y comparar el desempeño de los algoritmos MA y BVNS propuestos en esta investigación, así como de la heurística MACH, se han elegido las siguientes métricas estándares utilizadas en la literatura:

- $Best$: Calidad de la mejor solución, es decir, el menor $CBS(G, \varphi)$ encontrado por cada algoritmo.

- σ: Desviación estándar del $CBS(G, \varphi)$.

- hit: Número de veces que se alcanzó la mejor solución reportada por un algoritmo.

- τ: Tiempo promedio en segundos en el que se alcanzó la mejor solución.

- $RMSE$: Error Cuadrático Medio Relativo. El $RMSE$ es utilizado para medir la diferencia entre los valores esperados de un modelo y los valores que dicho modelo produce en realidad. En optimización el $RMSE$ puede ser empleado para evaluar el desempeño de un algoritmo al medir las diferencias entre las soluciones producidas por éste (Y_i) con respecto a la mejor solución conocida ($\hat{Y}$) para una instancia I. Para un desempeño perfecto $\hat{Y} = \hat{Y}_i$ y $RMSE = 0$. El $RMSE$ puede tomar valores entre cero e infinito, con cero correspondiendo al caso ideal. El $RMSE$ puede ser definido formalmente usando la Ecuación 4.1, donde R se refiere al número de ejecuciones del algoritmo.

$$RMSE(I) = \sqrt{\frac{\sum_{i=1}^{R} (\frac{Y_i - \hat{Y}}{\hat{Y}})^2}{R}}\,. \tag{4.1}$$

4.2 Instancias de prueba

Se utilizaron 412 grafos de prueba divididos en cuatro conjuntos de instancias, denominados: *Productos-Cartesianos*, *Harwell-Boeing*, *Estándar* y *Aleatorios*, los cuales han sido reportados en la literatura [19, 52]. Las características de cada uno de los conjuntos de prueba son descritas a continuación.

4.2.1 Instancias *Productos-Cartesianos*

Las instancias de este conjunto fueron producidas mediante el producto cartesiano de dos grafos $G_n \times H_m$. Ambos grafos pueden ser de tipo camino (P), ciclo (C) o grafo completo (K), con grado en el rango $3 \leq n, m \leq 9$, sujeto a las condiciones descritas en la Sección 2.4.2 referentes a las fórmulas para calcular cotas superiores. Dentro de este grupo existen los siguientes subconjuntos de instancias cuyo tamaño varia entre 9 y 81 vértices, y entre 18 y 648 aristas.

- $P \times P$: 28 grafos resultantes del producto cartesianos de dos caminos.
- $C \times C$: 28 grafos resultantes del producto cartesianos de dos ciclos.
- $K \times K$: 28 grafos resultantes del producto cartesianos de dos grafos completos.
- $P \times C$: 49 grafos resultantes del producto cartesiano de un grafo tipo camino y un grafo de tipo ciclo.
- $P \times K$: 49 grafos resultantes del producto cartesiano de un grafo tipo camino y un grafo completo.
- $C \times K$: 49 grafos resultantes del producto cartesiano de un grafo tipo ciclo y un grafo completo.

Para $P \times C$, $P \times K$ y $C \times K$ ocurre que $G_n \times H_m \neq H_m \times G_n$, por lo que estos subconjuntos poseen un número diferente de instancias que el resto.

4.2.2 Instancias *Harwell-Boeing*

Los grafos de este conjunto pertenecen a la colección de matrices dispersas Harwell-Boeing[1], las cuales modelan diversos problemas de ingeniería. Las 71 instancias seleccionadas están divididas en los siguientes subconjuntos:

[1] `http://math.nist.gov/MatrixMarket/data/Harwell-Boeing`

- can: 18 grafos de entre 24 y 1072 vértices.
- dwt: 24 grafos de entre 59 y 2680 vértices.
- $bcspwr$: 10 grafos de entre 39 y 5300 vértices.
- $mixed$: 19 grafos, de diversos tipos, de entre 9 y 989 vértices.

4.2.3 Instancias *Aleatorios*

Este conjunto está conformado por 50 grafos aleatorios generados mediante el método Erdös-Rényi, en el cual existe una arista entre cada par de vértices $u, v \in V$ con una probabilidad erp dada. En esta investigación para cada una de las probabilidades $erp \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ fueron creados 10 grafos con $n = 100$ vértices.

4.2.4 Instancias *Estándar*

Este conjunto está conformado por grafos con una estructura estándar definida, tales como: caminos, ciclos, ruedas, potencias de ciclos y bipartitas completos. El conjunto consta de un total de 60 instancias con $n \in \{100, 120, 140, 160, 180, 200, 400, 600, 800, 1000\}$ vértices, organizados de la siguiente manera:

- $path$: 10 grafos de tipo camino.
- $cycle$: 10 grafos de tipo ciclo.
- $wheel$: 10 grafos de tipo rueda.
- $cyclePow$: Un total de 20 grafos de tipo ciclo elevados a la potencia $k \in \{2, 10\}$, organizados de la siguiente forma:
  - $cyclePow_2$: 10 grafos C_n^k, $k = 2$.

- $cyclePow_{10}$: 10 grafos C_n^k, $k = 10$.

- $bipartiteComplete$: 10 grafos de tipo bipartita completo cuyos subconjuntos tienen cada uno $\frac{n}{2}$ vértices.

4.3 Condiciones experimentales

Los algoritmos MA y BVNS fueron codificados en lenguaje C++ versión 4.9.2 y compilados con g++ utilizando la bandera de optimización -O3. Hamon *et al.* [19] proporcionaron el código fuente de la heurística MACH implementada en Python 2.7.

Toda la experimentación fue realizada en el cluster *Neptuno* para cómputo de alto rendimiento del CINVESTAV-Tamaulipas, el cual cuenta con 104 nodos de procesamiento equipados con procesadores Intel(R) Xeon X5550 2.6 Hz y 16 GB de RAM.

En razón de la naturaleza no determinista de los algoritmos evaluados fueron realizadas 31 ejecuciones secuenciales de los métodos MACH, MA y BVNS para cada una de las 412 instancias. En cada ocasión se fijó un tiempo máximo de ejecución de 300 segundos para los algoritmos MA y BVNS.

4.4 Parámetros de los algoritmos propuestos

Para el Algoritmo Memético diseñado, los parámetros utilizados son:

- μ: El número de individuos en la población.

- CP: Probabilidad de cruza para cada pareja de individuos padre.

- MP: Probabilidad de mutación, para cada individuo descendiente.

- IP: Probabilidad de inversión para cada individuo superviviente.

- $maxG$: Número máximo de generaciones.
- $maxT$: Tiempo máximo de ejecución, en segundos.

La sintonización de los parámetros μ, CP, MP y IP fue realizada de manera automática mediante el paquete *irace* del lenguaje R [38]. Los parámetros $maxG$ y $maxT$ fueron fijados manualmente, como se muestra en la Tabla 4.1, con la finalidad de garantizar condiciones de comparación equitativas para los algoritmos MA y BVNS.

Tabla 4.1: Parámetros del algoritmo MA.

Parámetro	Rango de sintonización	Valor final
μ	$[10, 100]$	20
CP	$0.75, 0.95]$	0.780
MP	$[0.050, 0.25]$	0.143
IP	$0.050, 0.25]$	0.240
$maxG$	-	10,000,000
$maxT$	-	300

Por su parte, el algoritmo BVNS emplea como parámetros únicamente los valores de sus criterios de paro: número máximo de iteraciones y tiempo máximo de ejecución en segundos.

4.5 Resultados experimentales

En esta sección se muestra un resumen comparativo del desempeño de los algoritmos MACH, MA y BVNS con inicialización avara (BVNS+Greedy). Para cada subconjunto de instancias se presenta el promedio del mejor CBS encontrado ($Best$), el promedio de la desviación estándar (σ), el promedio del número de veces que el algoritmo alcanzó su mejor solución (hit), el promedio del Error Cuadrático Medio Relativo ($RMSE$) y el tiempo promedio en segundos (τ) en el que cada algoritmo alcanzó su mejor solución.[2]

[2]Las instancias de prueba, así como los resultados detallados de los experimentos presentados en este capítulo pueden ser consultados en línea en la dirección: `http://www.tamps.cinvestav.mx/~ertello/cbsp.php`.

4.5.1 Resultados para el conjunto *Productos-Cartesianos*

En la Tabla 4.2 se muestra el desempeño de los algoritmos MACH, MA y BVNS+Greedy según las métricas establecidas en la Sección 4.1, para el grupo de instancias *Productos-Cartesianos*. Se observa que los algoritmos MA y BVNS+Greedy desarrollados en esta investigación presentan entre sí un desempeño bastante similar, especialmente en términos de costo promedio de la mejor solución, con apenas 0.08 unidades de diferencia.

En este aspecto MA y BVNS+Greedy mejoran el desempeño de MACH en aproximadamente 41 % y 12 %, en los subconjuntos de instancias $P \times P$ y $K \times K$, y en menor medida en el resto de subconjuntos. El algoritmo BVNS+Greedy demostró ser el más consistente, presentando la menor desviación estándar en promedio, además de producir la mejor solución conocida en casi la totalidad de las ocasiones, como se observa en las columnas hit y $RMSE$. Por su parte, MA alcanza la mejor solución conocida con un margen de error y una desviación estándar promedio superiores a los presentados por BVNS+Greedy pero inferiores a los de MACH.

Para este grupo de instancias se observa que el algoritmo BVNS+Greedy presenta el mejor desempeño según las métricas, con excepción del tiempo τ, en el que MACH sobresale dada la naturaleza estructurada de las instancias. Dentro de los algoritmos comparados, MA es visiblemente

Tabla 4.2: Resumen del desempeño de MACH, MA y BVNS+Greedy, para el conjunto *Productos-Cartesianos*.

	mach					MA					BVNS-Greedy				
Instancias	$Best$	σ	hit	$RMSE$	τ	$Best$	σ	hit	$RMSE$	τ	$Best$	σ	hit	$RMSE$	τ
$C \times C$	292.29	39.19	8.46	1.27	0.00	271.39	0.03	30.96	0.00	4.83	271.39	0.00	31.00	0.00	0.04
$P \times P$	290.04	40.47	7.71	4.33	0.00	171.14	21.30	15.89	0.47	23.83	170.68	0.02	30.89	0.00	1.76
$P \times C$	258.12	2.06	24.71	0.95	0.00	226.90	10.65	27.45	0.08	11.34	226.90	0.18	30.22	0.00	0.66
$K \times K$	1889.29	13.80	2.54	0.66	0.00	1661.14	0.00	31.00	0.00	0.31	1661.14	0.00	31.00	0.00	0.02
$P \times K$	480.73	0.33	27.00	0.03	0.00	480.59	0.00	31.00	0.00	0.26	480.59	0.00	31.00	0.00	0.03
$C \times K$	520.98	0.33	27.43	0.04	0.00	519.92	0.00	31.00	0.00	0.16	519.92	0.00	31.00	0.00	0.04
Promedio	621.91	16.03	16.31	1.21	0.00	555.18	5.33	27.88	0.09	6.79	555.10	0.03	30.85	0.00	0.42

el más demandante en términos de tiempo, especialmente con el subconjunto $P \times P$.

4.5.2 Resultados para el conjunto *Harwell-Boeing*

El desempeño de los algoritmos MACH, MA y BVNS+Greedy para el conjunto *Harwell-Boeing* es ilustrado por la Tabla 4.3, en la que se observa que el algoritmo BVNS+Greedy presenta el mejor desempeño promedio considerando la totalidad de las instancias del conjunto, con alrededor de 31 % de mejora respecto a MACH. Para cada subconjunto, BVNS+Greedy es el algoritmo que proporciona la mejor solución conocida con mayor regularidad, en especial con los subconjuntos can y $mixed$, como se aprecia por sus valores de $Best$, hit y $RMSE$. El costo de las soluciones de MA reduce lo reportado por MACH en los conjuntos can, dwt y $mixed$ siendo inferior aproximadamente por 23 %, 34 % y 24 % respectivamente, mientras que se incrementa 365 % con el conjunto $bcspwr$, siendo éste en el que se presenta el mayor error $RMSE$. En términos de tiempo, MACH resultó ser capaz de construir una solución en un tiempo inferior al empleado por MA y BVNS+Greedy para los subconjuntos can y $mixed$. Sin embargo, es notorio el alto tiempo promedio que MACH demora en construir una solución para las instancias del subconjunto $bcspwr$, más de siete veces superior al requerido por BVNS+Greedy. En términos generales, BVNS+Greedy produjo las soluciones menos costosas en un tiempo apenas superior al empleado por MA por menos de medio minuto, además de proporcionar con mayor frecuencia la mejor solución conocida para todos los subconjuntos de grafos *Harwell-Boeing*.

4.5.3 Resultados para el conjunto *Aleatorios*

En la Tabla 4.4 se presentan los resultados de los algoritmos MACH, MA y BVNS+Greedy, promediando las métricas para el grupo de instancias aleatorias. Para este grupo de instancias MA y BVNS+Greedy muestran un desempeño cercano entre sí (menos de cien unidades de diferencia en promedio) en términos de costo promedio de la mejor solución , si bien los resultados de

Tabla 4.3: Resumen del desempeño de MACH, MA y BVNS+Greedy, para el conjunto *Harwell-Boeing*.

	mach					MA					BVNS-Greedy				
Instancias	*Best*	σ	*hit*	*RMSE*	τ	*Best*	σ	*hit*	*RMSE*	τ	*Best*	σ	*hit*	*RMSE*	τ
can	63434.67	10705.99	1.11	4.21	41.20	48400.00	2623.52	5.83	1.13	104.34	40173.83	3647.28	10.67	0.28	144.81
bcspwr	65764.00	8648.55	1.00	2.53	1520.04	240039.80	16911.04	1.10	5.72	184.07	56528.00	8289.35	5.70	0.98	196.80
dwt	62651.75	9935.17	2.25	4.84	224.14	40808.08	7564.46	2.25	2.30	145.73	39737.08	8507.82	5.83	0.98	196.60
mixed	41069.06	2712.32	6.06	3.72	15.42	31042.78	1033.46	9.50	3.22	101.29	24077.11	1885.28	15.94	0.67	102.52
Promedio	58229.87	8000.51	2.60	3.83	450.20	90072.67	7033.12	4.67	3.09	133.86	40129.01	5582.43	9.54	0.73	160.18

BVNS+Greedy son mejores, ambos algoritmos mejoran lo reportado por MACH, especialmente en el subconjunto $random01$. MA y BVNS+Greedy también son capaces de alcanzar la mejor solución conocida con mayor frecuencia, como se observa en su $RSME$ por debajo de 0.1 en la mayoría de los casos. Por su parte, el algoritmo MACH reporta soluciones en promedio más costosas en un tiempo menor al empleado tanto por MA como por BVNS+Greedy.

Tabla 4.4: Resumen del desempeño de MACH, MA y BVNS+Greedy, para el conjunto *Aleatorios*.

	mach					MA					BVNS-Greedy				
Instancias	*Best*	σ	*hit*	*RMSE*	τ	*Best*	σ	*hit*	*RMSE*	τ	*Best*	σ	*hit*	*RMSE*	τ
random01	9370.10	226.62	5.20	1.80	0.10	7357.60	91.37	1.00	0.10	186.10	7344.50	18.24	2.50	0.02	177.64
random03	34958.80	107.32	21.70	1.07	0.30	29479.20	174.04	1.00	0.07	122.15	29417.50	38.27	2.20	0.01	183.72
random05	62241.80	184.92	24.10	0.89	0.34	53929.10	223.38	1.00	0.05	78.86	53822.00	39.08	1.30	0.00	194.49
random07	88141.40	214.69	12.10	0.63	0.51	79617.40	174.40	1.00	0.03	51.99	79505.90	40.22	1.40	0.00	194.04
random09	113674.00	294.30	1.70	0.37	0.66	107060.00	128.00	1.00	0.01	82.57	107007.00	32.56	1.60	0.00	194.77
Promedio	61677.22	205.57	12.96	0.95	0.38	55488.66	158.24	1.00	0.05	104.33	55419.38	33.67	1.80	0.01	188.93

4.5.4 Resultados para el conjunto *Estándar*

La Tabla 4.5 contiene los resultados promedio de MACH, MA y BVNS+Greedy para el grupo de instancias *Estándar* conformado por grafos con estructura específica. Como se esperaba, dado que MACH es una heurística especialmente diseñada para construir una solución explorando la estructura del grafo, es precisamente en este grupo de instancias donde es capaz de alcanzar el mejor desempeño,

llegando al óptimo en la totalidad de las ocasiones. El tiempo τ empleado por MACH para producir dicha solución crece considerablemente con el subconjunto pow en relación al resto de subconjuntos, 500 veces superior al empleado con el subconjunto $wheel$. Por su parte, MA y BVNS+Greedy presentan el mayor error $RMSE$ en este grupo de instancias, con excepción de los subconjuntos pow y $bipartite$. El tiempo τ empleado por MA y BVNS+Greedy es alto en comparación con MACH, en los subconjuntos $path$, $cycle$ y $wheel$; sin embargo en promedio resulta ser de menos del 50 % del tiempo requerido por MACH.

Tabla 4.5: Resumen del desempeño de MACH, MA y BVNS+Greedy, para el conjunto *Estándar*.

	mach					MA					BVNS-Greedy				
Instancias	$Best$	σ	hit	$RMSE$	τ	$Best$	σ	hit	$RMSE$	τ	$Best$	σ	hit	$RMSE$	τ
path	369.00	0.00	31.00	0.00	0.01	670.70	175.09	14.40	4.57	146.09	387.10	110.77	5.80	2.80	69.35
cycle	370.00	0.00	31.00	0.00	0.01	700.00	171.87	20.10	4.55	121.18	677.40	69.71	2.90	5.48	61.98
wheel	57920.00	0.00	31.00	0.00	1.32	58242.80	167.48	19.60	0.03	125.73	59525.80	280.05	4.90	0.16	168.83
pow	22692500.00	0.00	31.00	0.00	679.68	22692500.00	0.00	31.00	0.00	0.02	22692500.00	0.00	31.00	0.00	0.00
bipartite	11346250.00	0.00	31.00	0.00	172.21	11346250.00	0.00	31.00	0.00	20.74	11346250.00	0.00	31.00	0.00	13.02
Promedio	6819481.80	0.00	31.00	0.00	170.65	6819672.70	102.89	23.22	1.83	82.75	6819868.06	92.11	15.12	1.69	62.63

La Figura 4.1 muestra la distribución del promedio de error, representado como gráficas de caja. Para una mejor visualización se ha dividido el conjunto de instancias en los cuatro subconjuntos mencionados con anterioridad y se han retirado los valores atípicos.

Como se aprecia en la Figura 4.1, para todos los conjuntos de instancias, con excepción de los grafos *Estándar*, BVNS+Greedy proporciona los resultados con menor tasa de error respecto al menor costo de solución alcanzado durante la experimentación con MACH, MA y BVNS+Greedy aquí reportada. Sin embargo, para dos de los cinco subconjuntos (pow y $bipartite$) del conjunto *Estándar*, MA y BVNS+Greedy alcanzan resultados con $RMSE = 0$ en un tiempo promedio menor al de MACH.

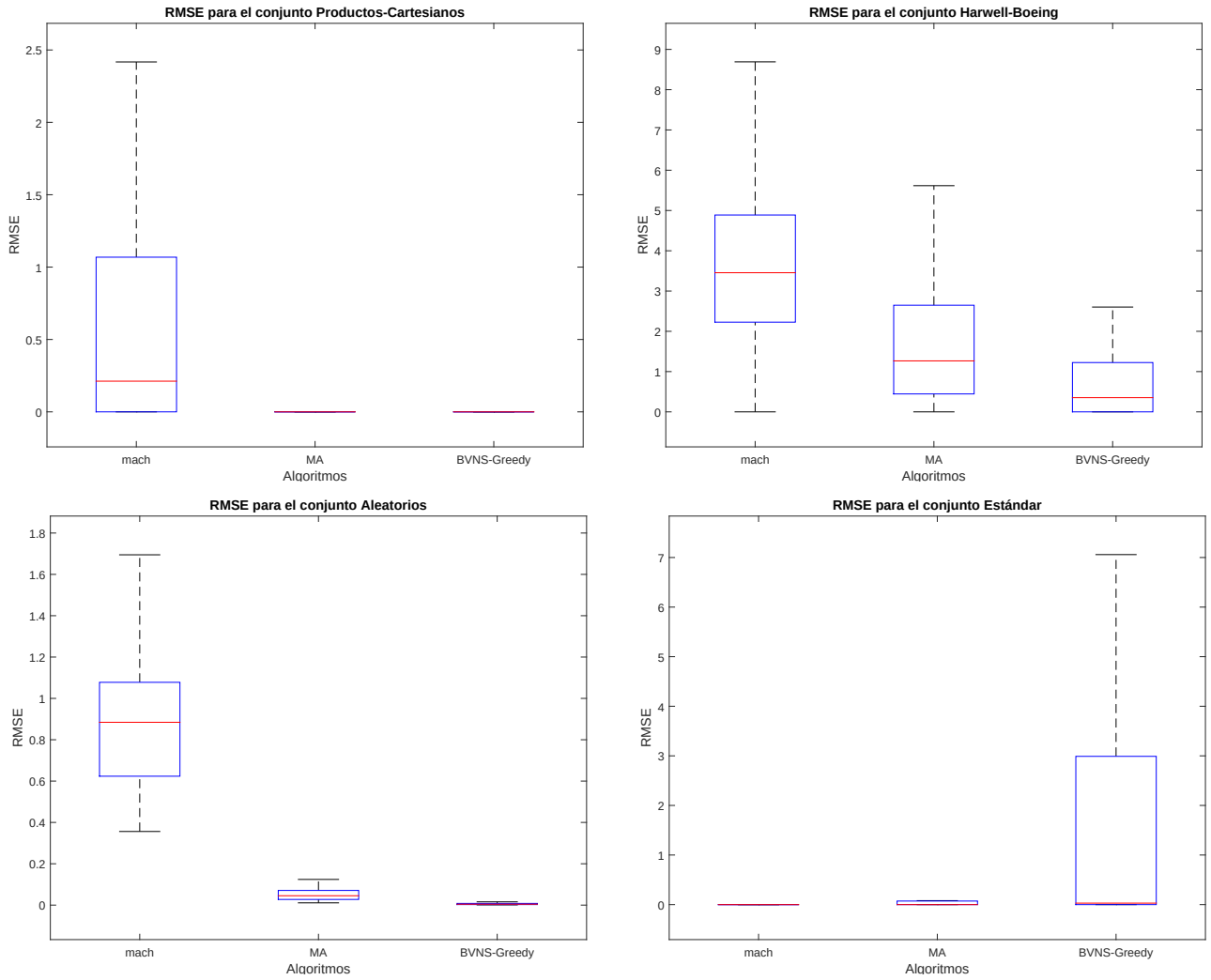


Figura 4.1: Distribución del promedio del $RMSE$ de MACH, MA y BVNS+Greedy, por grupo de instancias.

En lo referente a tiempo de ejecución, los algoritmos MA y BVNS+Greedy demostraron requerir, en la mayoría de los casos, mayor tiempo de cómputo que el método MACH. Sin embargo, esto se ve compensado por un menor costo de solución reflejado en un error menor con MA y especialmente con BVNS+Greedy. Si bien el algoritmo MA presenta un promedio de tiempo de ejecución menor que BVNS+Greedy para los grupos de instancias *Productos-Cartesianos*, *Harwell-Boeing* y *Aleatorios* según la Figura 4.2, al comparar sus respectivos promedios de error se observa que BVNS+Greedy fue capaz de encontrar soluciones de menor costo que MA y que MACH.

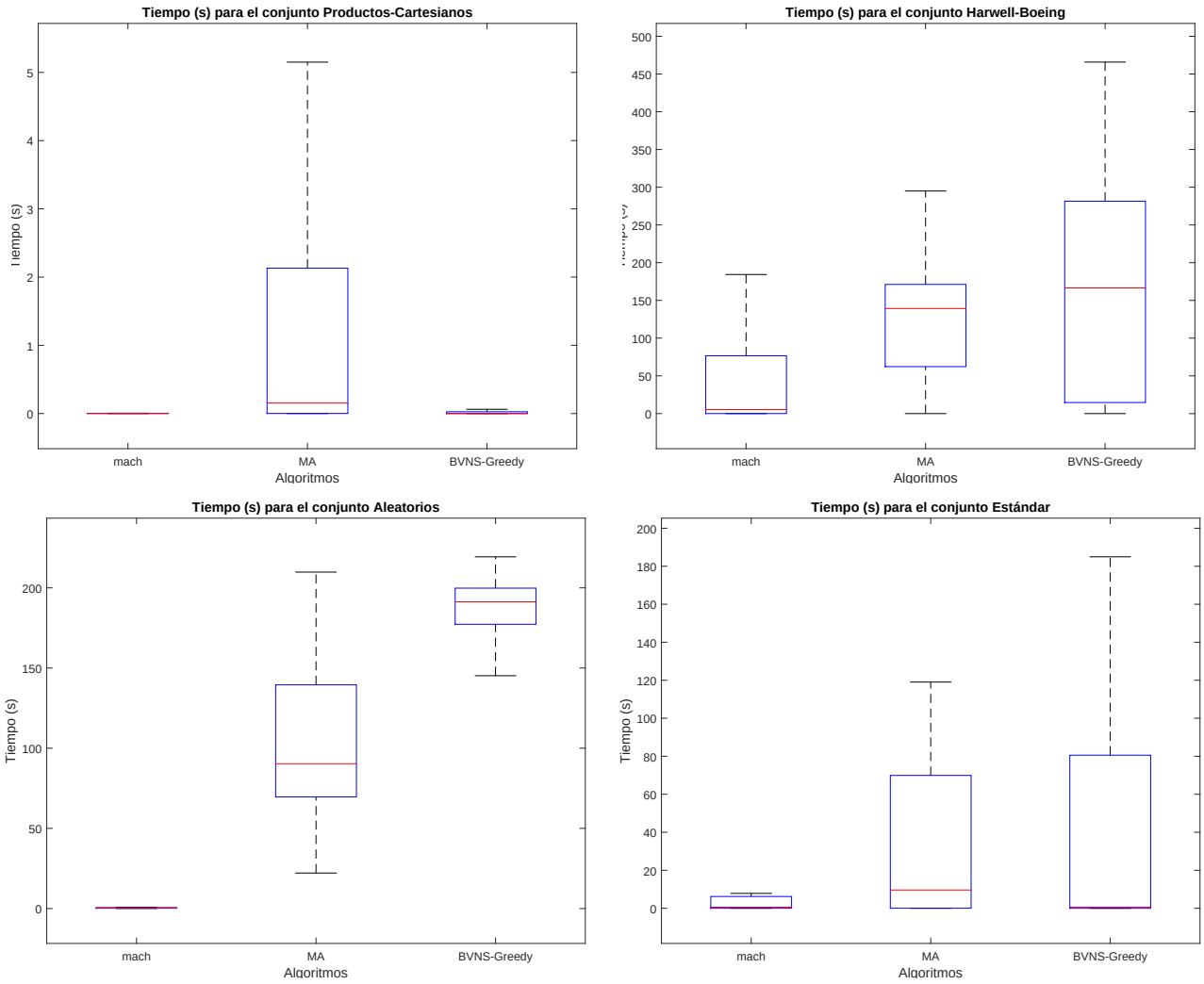


Figura 4.2: Distribución del promedio del tiempo de ejecución de MACH, MA y BVNS+Greedy, por grupo de instancias.

4.5.5 *Time-to-target*

En la literatura se ha propuesto el uso de la distribución del tiempo de ejecución requerido por un algoritmo para alcanzar una solución de un costo determinado, denominado *time-to-target*, como una alternativa para medir y comparar el desempeño respecto a otros algoritmos. Este enfoque se basa en la conjetura de Hoos and Stützle [25, 26] que indica que el tiempo de ejecución de los algoritmos de búsqueda local para optimización combinatoria sigue una distribución exponencial.

Las gráficas de *time-to-target*, utilizadas por primera vez por Feo *et al.* [12], han sido recomendadas por Hoos and Stützle [22, 24] y empleadas repetidamente por Lozano *et al.* [39] y Duarte *et al.* [10] en la comparación de algoritmos de búsqueda local para el Problema del Anti Ancho de Banda.

En esta investigación se hizo uso de los código proporcionados por Aiex *et al.* [2] para generar gráficas del *time-to-target* y para determinar la probabilidad de que un algoritmo alcance un costo objetivo determinado en un tiempo menor que otro. Para este experimento se presenta la instancia can_838 del subconjunto can, con $n = 838$ vértices, dado que para el resto de las instancias se observa un comportamiento similar. Como costo objetivo se utilizó el mejor costo de solución reportado por MACH de entre un total de 50 ejecuciones.

En Tabla 4.6 cada celda registra la probabilidad de que un algoritmo dado (filas) produzca una solución de costo menor o igual al alcanzado por el resto de algoritmos (distribuidos en las columnas) en un tiempo menor o igual, para la instancia can_838. El algoritmo MA presenta, al compararse con el resto de algoritmos, la mayor probabilidad de alcanzar el costo objetivo en un tiempo menor o igual. BVNS con inicialización aleatoria (BVNS+Random) se muestra competitivo respecto a BVNS+Greedy. Es destacable que para esta instancia, BVNS+MACH no tiene posibilidades de alcanzar el costo objetivo dado el tiempo notoriamente superior que requiere MACH para crear la solución inicial.

Tabla 4.6: Probabilidad de que un algoritmo (filas) produzca una solución de costo menor o igual al costo objetivo en un tiempo menor o igual a otro algoritmo (columnas), para la instancia can_838.

Algoritmos	MA	BVNS+Random	BVNS+Greedy	BVNS+mach
MA		1.0000	1.0000	1.0000
BVNS+Random	0.0000		0.4226	1.0000
BVNS+Greedy	0.0000	0.5774		1.0000
BVNS+mach	0.0000	0.0000	0.0000	

En la Figura 4.3 se muestra la probabilidad acumulada de que, dado un tiempo de ejecución, los algoritmos implementados en esta investigación alcancen una solución del costo objetivo fijado para la instancia can_838. Cada algoritmo fue ejecutado 50 veces, almacenado en cada ejecución el tiempo en segundos en el que se alcanzó una solución con costo menor o igual al costo objetivo (de 256480). Como se observa en dicha figura, el algoritmo BVNS+MACH requiere un tiempo considerablemente mayor.

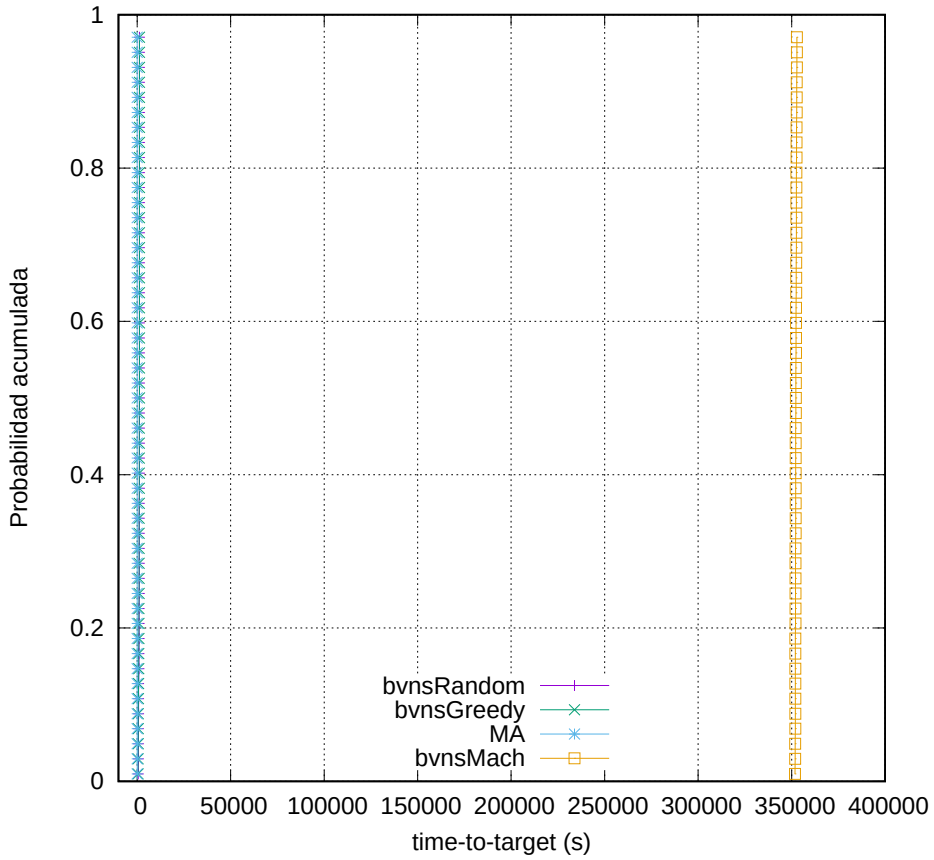


Figura 4.3: Gráfica del *time-to-target* para los algoritmos MA y BVNS con los tres tipos de inicialización.

En la Figura 4.4 es posible apreciar con mayor detalle el comportamiento de la probabilidad acumulada según el tiempo de ejecución para los algoritmos MA, BVNS+Greedy y BVNS+Random. Se observa que BVNS, en ambas versiones de inicialización, es capaz de alcanzar el costo objetivo,

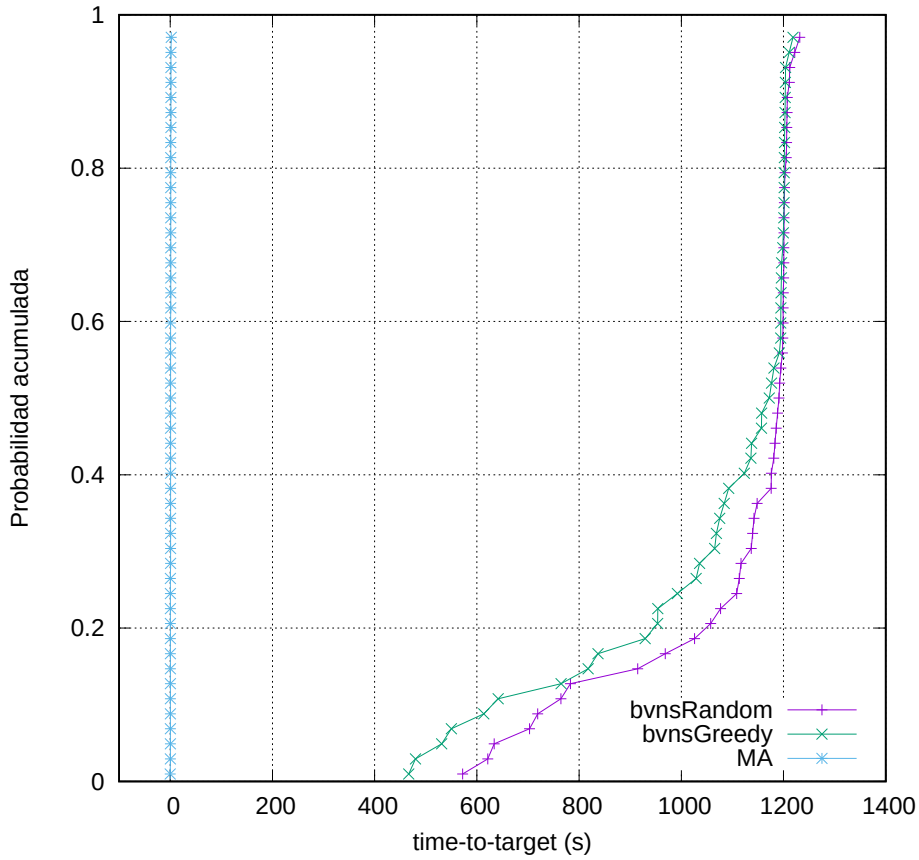


Figura 4.4: Gráfica del *time-to-target* para los algoritmos MA y BVNS con inicialización avara y aleatoria.

con 100 % de probabilidad a partir de aproximadamente 1200 segundos de ejecución. Por su parte MA requiere menos de 10 segundos para alcanzar el costo objetivo con el 100 % de probabilidad. Sin embargo, a pesar de que para la instancia can_835 MA es capaz de llegar al costo objetivo reportado por MACH en un tiempo menor que el requerido por cualquiera de las tres variantes de BVNS, el $RMSE$ promedio de MA para las instancias del subconjunto can es aproximadamente cuatro veces superior al de BVNS+Greedy, según la Tabla 4.3. De lo anterior se deduce que aunque, para la instancia can_838, MA es capaz de alcanzar una mejor solución que BVNS+Greedy en el corto plazo, al incrementar los tiempos de ejecución de ambos algoritmos BVNS+Greedy reporta una solución mejor que la de MA. Por tanto, BVNS+Greedy tiene la capacidad de conducir el proceso de búsqueda hacia mejores soluciones en el largo plazo.

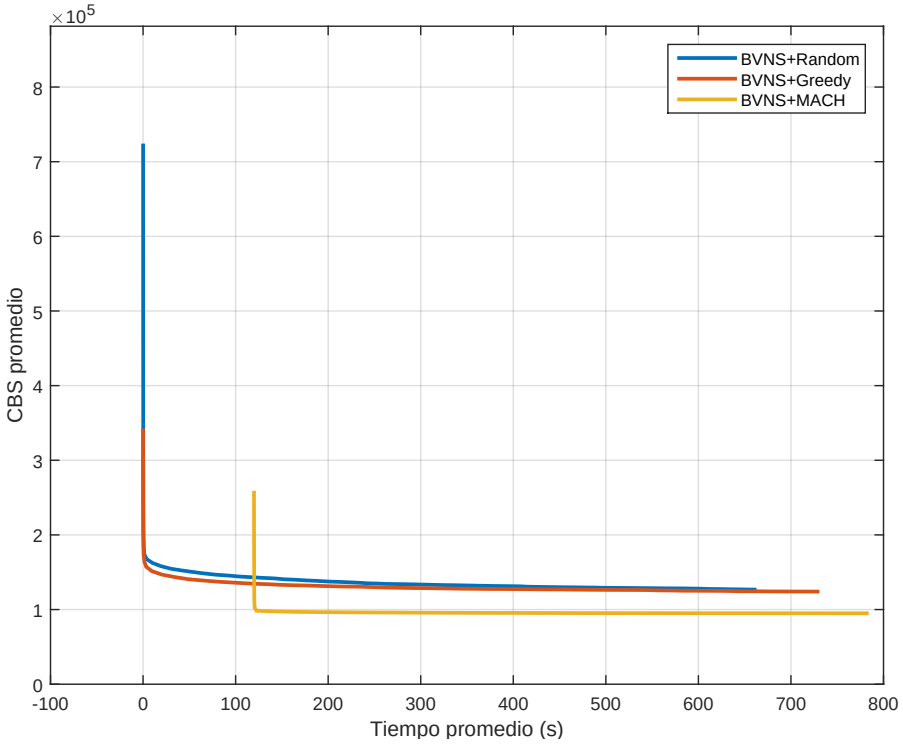


Figura 4.5: Convergencia de BVNS según el método de construcción de la solución inicial.

4.5.6 Alternativas de inicialización para BVNS

En la Sección 3.2.1 fueron descritas tres variantes para la construcción de la solución inicial del algoritmo BVNS, las cuales son: aleatoria por el método Fisher-Yates [13], algoritmo avaro y MACH. En esta sección se compara el desempeño de BVNS según la alternativa de inicialización utilizada. El objetivo es comprobar si, en efecto, la inicialización avara aporta ventajas, ya sea en cuanto costo de solución o tiempo de ejecución, respecto a la inicialización aleatoria, y además, evaluar el impacto en el desempeño al utilizar MACH como método de inicialización para BVNS.

La Figura 4.5 muestra la convergencia a través del tiempo para cada uno de los tres métodos de inicialización para BVNS para la instancia can_838 del conjunto *Harwell-Boeing*. De acuerdo a lo esperado, la construcción de la solución mediante un algoritmo avaro no sólo permite arrancar la búsqueda desde un punto con menor costo si no que también presenta una convergencia más

rápida y constante a lo largo del tiempo comparada con la inicializadción aleatoria. Por otro lado la inicialización mediante MACH aporta una solución inicial de menor costo que la inicialización avara y permite llegar a una mejor solución final. Sin embargo, al utilizar este método, debe tomarse en cuenta que el tiempo requerido para construir la solución inicial mediante MACH puede demandar un tiempo considerable.

Dicho comportamiento puede ser apreciado en las Tablas 4.7, 4.8, 4.9 y 4.10, particularmente en la columna τ correspondiente al tiempo promedio de ejecución. Para el caso del algoritmo BVNS+MACH, τ equivale a la suma del tiempo promedio de MACH y BVNS.

En el caso del conjunto de instancias *Productos-Cartesianos*, cuyos resultados se resumen en la Tabla 4.7, las tres alternativas de inicialización son capaces de alcanzar un desempeño perfecto respecto a la mejor solución, con un $RMSE = 0$ en todos los casos. La inicialización aleatoria es la que requiere un menor tiempo de ejecución promedio τ.

Tabla 4.7: Resumen del desempeño de las tres alternativas de inicialización de BVNS para el conjunto *Productos-Cartesianos*.

	BVNS-Random					BVNS-Greedy					BVNS-mach				
Instancias	$Best$	σ	hit	$RMSE$	τ	$Best$	σ	hit	$RMSE$	τ	$Best$	σ	hit	$RMSE$	τ
$C \times C$	271.39	0.00	31.00	0.00	0.04	271.39	0.00	31.00	0.00	0.04	271.39	0.00	31.00	0.00	0.00
$P \times P$	170.68	0.03	30.57	0.00	1.56	170.68	0.02	30.89	0.00	1.76	170.68	0.02	30.75	0.00	1.74
$P \times C$	226.90	0.16	30.47	0.00	0.58	226.90	0.18	30.22	0.00	0.66	226.90	0.14	29.80	0.00	0.85
$K \times K$	1661.14	0.00	31.00	0.00	0.02	1661.14	0.00	31.00	0.00	0.02	1661.14	0.00	31.00	0.00	0.02
$P \times K$	480.59	0.00	31.00	0.00	0.05	480.59	0.00	31.00	0.00	0.03	480.59	0.00	31.00	0.00	0.00
$C \times K$	519.92	0.00	31.00	0.00	0.04	519.92	0.00	31.00	0.00	0.04	519.92	0.00	31.00	0.00	0.00
Promedio	555.10	0.03	30.84	0.00	0.38	555.10	0.03	30.85	0.00	0.42	555.10	0.03	30.76	0.00	0.44

En cuanto al conjunto *Harwell-Boeing* (ver Tabla 4.8), tanto en el costo promedio de la mejor solución $Best$ como en $RMSE$ el método avaro presenta mejores resultados que el aleatorio, en un tiempo de ejecución promedio similar, con menos de 10 segundos de diferencia. Si bien el menor $RMSE$ para este grupo de instancias corresponde a BVNS+MACH, también ocurre que el tiempo de ejecución promedio τ es el mayor. Especialmente para el subconjunto $bcspwr$, donde el tiempo de ejecución promedio es más de ocho veces superior al de BVNS+Greedy.

Tabla 4.8: Resumen del desempeño de las tres alternativas de inicialización de BVNS para el conjunto *Harwell-Boeing*.

	BVNS-Random					BVNS-Greedy					BVNS-mach				
Instancias	$Best$	σ	hit	$RMSE$	τ	$Best$	σ	hit	$RMSE$	τ	$Best$	σ	hit	$RMSE$	τ
can	37586.17	5196.49	10.89	0.59	144.87	40173.83	3647.28	10.67	0.55	144.81	34981.44	229.44	10.94	0.05	172.64
bcspwr	169594.40	8660.61	6.00	6.18	193.91	56528.00	8289.35	5.70	1.12	196.80	53938.30	561.84	7.10	0.11	1697.16
dwt	57288.33	10763.31	5.79	2.38	204.58	39737.08	8507.82	5.83	1.04	196.60	42531.83	370.72	6.54	0.27	401.03
mixed	24556.00	2241.40	15.94	1.29	113.47	24077.11	1885.28	15.94	0.91	102.52	24136.33	348.83	16.06	0.15	90.55
Promedio	72256.22	6715.45	9.66	2.61	164.21	40129.01	5582.43	9.54	0.91	160.18	38896.98	377.71	10.16	0.14	590.35

Dada la poca estructura de los grafos pertenecientes al conjunto *Aleatorio*, ninguno de los tres métodos de inicialización se distingue particularmente sobre los demás, como se aprecia en la Tabla 4.9. En la totalidad de las métricas consideradas los tres métodos comparados presentan un comportamiento similar, en especial un $RMSE = 0.01$ en todos los casos, y un tiempo de ejecución promedio con menos de tres segundos de diferencia. De lo anterior se deduce que ni la incialización avara, ni la inicialización mediante MACH son capaces de proporcionar una solución inicial ventajosa respecto a una construcción aleatoria, para este tipo de instancias.

Tabla 4.9: Resumen de las tres alternativas de inicialización de BVNS para el conjunto *Aleatorios*.

	BVNS-Random					BVNS-Greedy					BVNS-mach				
Instancias	$Best$	σ	hit	$RMSE$	τ	$Best$	σ	hit	$RMSE$	τ	$Best$	σ	hit	$RMSE$	τ
random01	7344.20	20.00	2.50	0.02	169.45	7344.50	18.24	2.50	0.02	177.64	7344.50	16.51	2.70	0.01	167.20
random03	29417.60	38.08	2.80	0.01	185.08	29417.50	38.27	2.20	0.01	183.72	29417.90	37.36	3.00	0.01	183.61
random05	53822.50	40.84	1.70	0.00	185.95	53822.00	39.08	1.30	0.00	194.49	53823.60	39.98	1.60	0.00	187.39
random07	79505.80	39.36	1.00	0.00	193.64	79505.90	40.22	1.40	0.00	194.04	79505.60	43.58	1.20	0.00	198.40
random09	107011.00	36.20	1.10	0.00	198.25	107007.00	32.56	1.60	0.00	194.77	107006.00	32.36	1.10	0.00	193.73
Promedio	55420.22	34.89	1.82	0.01	186.47	55419.38	33.67	1.80	0.01	188.93	55419.52	33.96	1.92	0.01	186.07

Como se observó con anterioridad en el Tabla 4.5, el desempeño de MACH para el conjunto de instancias *Estándar* en lo referente a $RMSE$ es perfecto, con $RMSE = 0$, alcanzado el óptimo en todos los casos, dado el alto nivel de estructura de este tipo de grafos. Sin embargo, el tiempo de ejecución para los subconjuntos pow y $bipartite$ crece considerablemente. Este mismo comportamiento se ve reflejado al utilizar MACH como medio de inicialización para BVNS, como se aprecia en la Tabla 4.10. Además, para estos dos subconjuntos, las inicializaciones avara y aleatoria son capaces de proporcionar resultados similares en términos de costo de solución, en un tiempo promedio reducido en más del 95 % respecto a MACH.

Tabla 4.10: Resumen del desempeño de las tres alternativas de inicialización de BVNS para el conjunto *Estándar*.

	BVNS-Random					BVNS-Greedy					BVNS-mach				
Instancias	$Best$	σ	hit	$RMSE$	τ	$Best$	σ	hit	$RMSE$	τ	$Best$	σ	hit	$RMSE$	τ
path	2198.70	352.43	1.60	20.92	163.66	387.10	110.77	5.80	2.80	69.35	369.00	0.00	31.00	0.00	0.01
cycle	2078.80	353.33	2.50	20.68	146.76	677.40	69.71	2.90	5.48	61.98	370.00	0.00	31.00	0.00	0.01
wheel	60100.20	336.21	2.50	0.21	189.61	59525.80	280.05	4.90	0.16	168.83	57920.00	0.00	31.00	0.00	1.32
pow	22692500.00	0.00	31.00	0.00	0.00	22692500.00	0.00	31.00	0.00	0.00	22692500.00	0.00	31.00	0.00	679.68
bipartite	11346250.00	0.00	31.00	0.00	13.44	11346250.00	0.00	31.00	0.00	13.02	11346250.00	0.00	31.00	0.00	172.21
Promedio	6820625.54	208.39	13.72	8.36	102.69	6819868.06	92.11	15.12	1.69	62.63	6819481.80	0.00	31.00	0.00	170.65

En la Figura 4.6 se muestra la distribución del $RMSE$ por conjunto de instancia de BVNS inicializado con los tres métodos mencionados: aleatorio, avaro y MACH. Como se señaló con anterioridad, en términos de $RMSE$, los tres métodos proporcionan resultados similares para los conjuntos de instancias *Productos-Cartesianos* y *Aleatorios*.

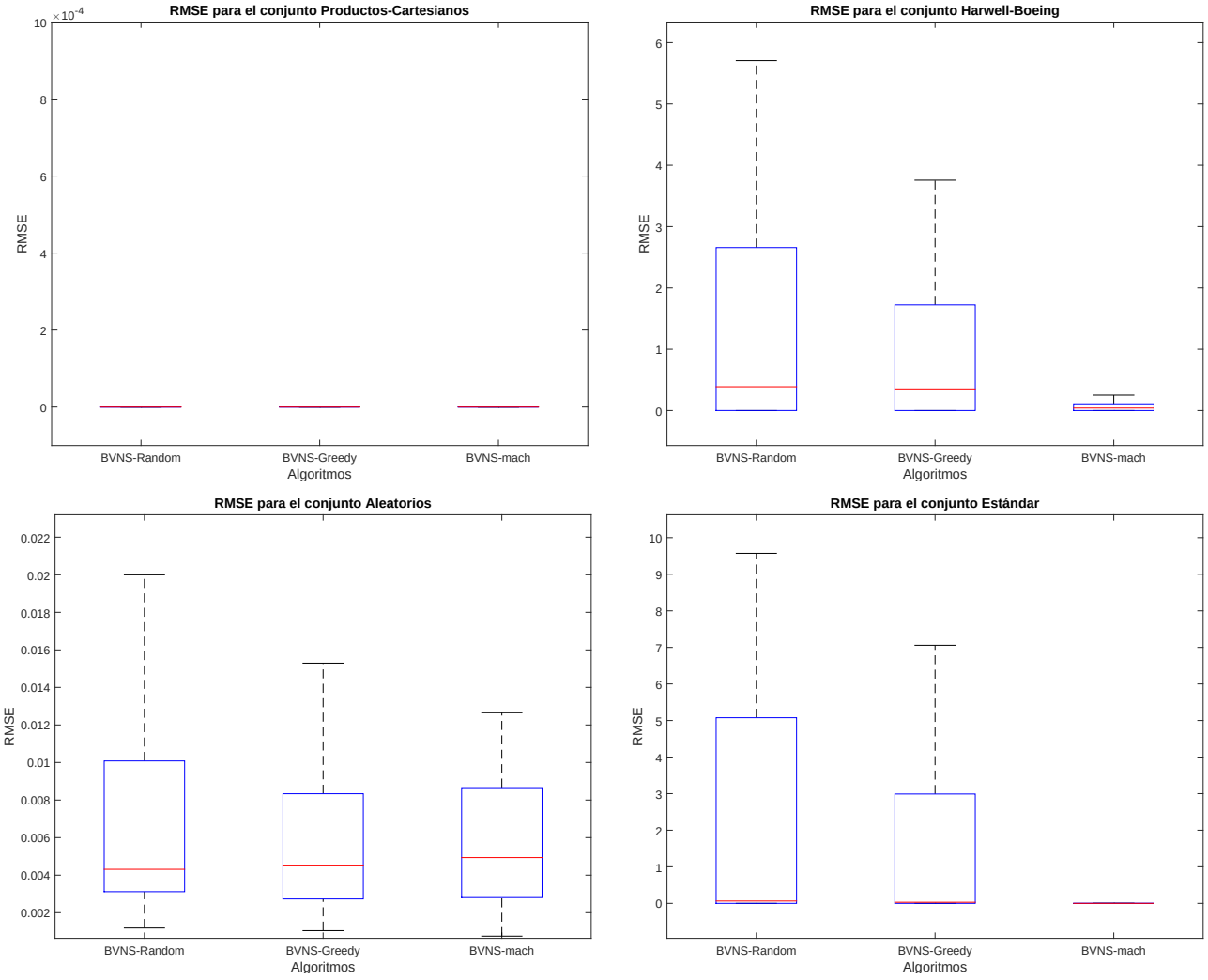


Figura 4.6: Distribución del promedio del $RMSE$ por grupo de instancias, para BVNS según el método de inicialización.

En la Figura 4.7 se presenta la distribución del tiempo de ejecución para cada una de la alternativas de inicialización. Se observa que, la inicialización mediante MACH sólo es capaz de resolver las instancias en un tiempo menor que BVNS con inicialización aleatoria o avara cuando dichas instancias

presentan una estructura definida. La alternativa de inicialización avara, para los conjuntos *Productos-Cartesianos*, *Harwell-Boeing* y *Estándar*, requiere en promedio un tiempo de ejecución menor que su contraparte aleatoria, y como se observó en la Figura 4.6, brinda menor tasa de error en la totalidad de instancias. De este modo se demuestra que, en general, mediante inicialización avara, BVNS es capaz de alcanzar una solución de menor costo y en menor tiempo que mediante la inicialización aleatoria.

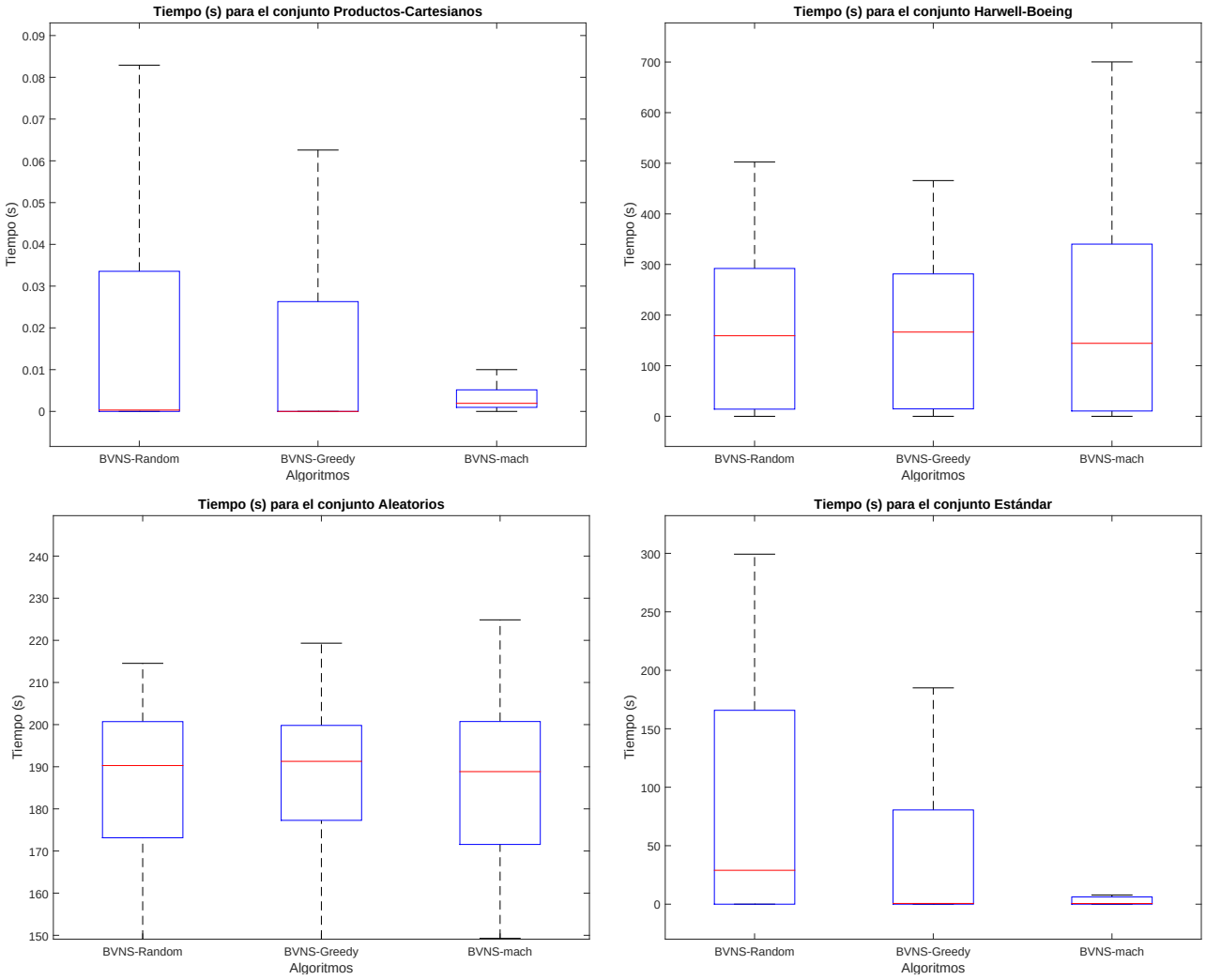


Figura 4.7: Distribución del tiempo promedio de ejecución por grupo de instancias, para BVNS según el método de inicialización.

4.6 Resumen del capítulo

En este capítulo se ha descrito la fase de experimentación realizada con los algoritmos propuestos, así como las condiciones bajo las que ésta se llevo a a cabo.

A partir de los resultados obtenidos, se notó que tanto el tiempo de ejecución de la heurística MACH como su tasa de error se ven altamente influenciados por el tipo de estructura de las instancias, más que por el tamaño de las mismas. De este modo, MACH sólo es capaz de encontrar soluciones cuyo costo iguale o mejore lo producido por BVNS y MA en instancias estructuradas, tales como los grafos del conjunto *Estándar*; e incluso en estos casos su tiempo de ejecución se ve incrementado en los subconjuntos pow y $bipartite$.

En términos de costo de solución, fue observado un desempeño favorable de los algoritmos MA y BVNS, mejorando los resultados obtenidos por MACH para los conjuntos de instancias *Productos-Cartesianos*, *Harwell-Boeing* y *Aleatorios* y para los subconjuntos pow y $bipartite$ del conjunto *Estándar*. BVNS demostró dominar al resto de algoritmos, presentando la menor tasa de error y un tiempo de ejecución moderado.

Al comparar las distintas alternativas para construir la solución inicial del algoritmo BVNS se observó que en general, BVNS+MACH demanda un tiempo de ejecución mayor, en varios casos considerable. Aunque el desempeño de esta alternativa en lo referente a calidad de solución es superior al de sus contrapartes aleatoria y avara, en los conjuntos *Productos-Cartesianos*, *Harwell-Boeing* y *Aleatorios*, así como en los subconjuntos pow y $bipartite$ del conjunto *Estándar*, dicha superioridad no compensa el gran incremento en el tiempo de ejecución requerido.

Por tanto, con base en los resultados observados, se considera que BVNS+Greedy ofrece el mejor balance entre calidad de solución y tiempo de ejecución.

5

Conclusiones y trabajo futuro

A lo largo de esta tesis se han presentado dos algoritmos metaheurísticos desarrollados en esta investigación para resolver el CBSP. Bajo el enfoque poblacional fue desarrollado un Algoritmo Memético [30, 42, 43] que permite combinar los operadores evolutivos de los Algoritmos Genéticos [9, 16, 21] con la Búsqueda Local [23, 32]. El Algoritmo Memético diseñado utiliza la selección por torneo binario, la cruza basada en el orden, una mutación heurística, superviviencia (μ, λ) y un operador de inversión. Los parámetros de este algoritmo fueron sintonizados utilizando la herramienta para sintonización automática *irace* [38] del lenguaje R.

Por otro lado, bajo el enfoque monosolución se diseñó e implementó una Búsqueda Básica por Vecindario Variable compuesta por cuatro fases: construcción de la solución inicial, perturbación, búsqueda local y cambio de vecindario. Se analizó el impacto de utilizar tres distintas alternativas para la creación de la solución inicial, empleando los métodos MACH, aleatorio y avaro. Con el propósito de disminuir el tiempo de ejecución demandado por el cálculo del costo de las soluciones potenciales, se desarrolló una función de evaluación incremental, la cual es aplicada en las implementaciones de

MA y BVNS propuestas.

La experimentación fue realizada con un total de 412 instancias de hasta $n = 5,300$ vértices, descritas a detalle en la Sección 4.2. Los resultados de dicha experimentación fueron discutidos en el Capítulo 4.

En éste capítulo se presentan las principales conclusiones generadas en esta investigación, se determina la veracidad de la hipótesis planteada en la Sección 1.2 y se evalúa el cumplimiento de los objetivos descritos en la Sección 1.3. Asimismo, se consideran algunos de los posibles temas de investigación futura derivados del estudio del problema CBSP y de las alternativas de solución propuestas en este documento.

5.1 Conclusiones

Con base en el estudio comparativo presentado en la Sección 4.5 se concluye que los algoritmos MA y BVNS implementados durante esta investigación son alternativas competitivas para solucionar el CBSP independientemente de la topología del grafo.

Del total de 412 instancias utilizadas, los algoritmos propuestos en ésta investigación reportaron 238 nuevas cotas inferiores. De estas cotas, 142 fueron alcanzadas por MA, mientras que BVNS, con inicialización avara, aleatoria y mediante MACH, lo logró en 203, 188 y 169 casos, respectivamente. MA igualó el desempeño de MACH, en términos de costo de solución para 165 de las instancias y lo mejoró para 231, restando así 16 instancias en las que MACH consigue mejores resultados que MA en 162.33 % en promedio.

Por su parte, BVNS+Greedy igualó 157 y mejoró 238 de los resultados reportados por MACH. Para las 17 instancias restantes las soluciones de BVNS+Greedy resultan más costosas que las de MACH por 3.36 % en promedio.

En este sentido, para 31 instancias los resultados de BVNS+Random fueron peores que los de MACH por un promedio de 69.40 %, sin embargo BVNS+Random fue capaz de igualar los resultados

obtenidos por *mach* para 152 instancias y de mejorarlos para 229.

Al aplicar BVNS a las soluciones producidas por MACH fue posible mejorar los resultados para 238 de las instancias. De las restantes 174 instancias, 60 corresponden al conjunto *Estándar* para el cuál MACH alcanza el óptimo en todos los casos. Por lo tanto, sólo 114 instancias, de las que se desconoce el valor exacto del óptimo, no pudieron ser mejoradas mediante BVNS+MACH.

El algoritmo MACH, por su enfoque centrado en la estructura del grafo, es generalmente rápido en grafos con estructura específica, como demuestran los resultados del conjunto *Estándar*, con excepción del subconjunto pow. Sin embargo, en grafos carentes de dicha estructura o con alto número de aristas su desempeño en términos de calidad de solución decae y se eleva el tiempo de cómputo. Esto se observó particularmente con el conjunto de grafos aleatorios, el subconjunto dwt de grafos *Harwell-Boeing* y el subconjunto pow de los grafos *Estándar*.

Los resultados indican que MA, y en mayor grado BVNS, presentan un buen desempeño con grafos generales, superando o igualando los resultados de MACH en la mayoría de los casos. Si bien el tiempo de cómputo requerido por MA y BVNS es en promedio superior al requerido por MACH este aspecto es compensado por la calidad de la solución y la consistencia de ésta a lo largo de diversas ejecuciones.

Por tanto, se ha respondido afirmativamente a la hipótesis formulada en esta investigación. El objetivo planteado de diseñar e implementar dos algoritmos competitivos, basados en metaheurísticas de tipo monosolución y poblacional se puede considerar cumplido. Las principales contribuciones obtenidas durante esta investigación son los algoritmos MA y BVNS, un *benchmark* compuesto por un conjunto de instancias diverso en el que se incluyen los resultados alcanzados por MACH, MA y BVNS, y las mejores cotas para dichas instancias.

5.2 Trabajo futuro

Dado que algoritmo BVNS demostró un desempeño superior al del Algorimo Memético se propone ampliar los conjuntos de instancias para incluir grafos de $n \leq 100,000$ vértices, y rediseñar la fase de exploración del algoritmo BVNS para que sea posible resolver este tipo de instancias reduciendo el número de soluciones vecinas visitadas y eligiendo dichas soluciones de manera inteligente.

Si bien el desempeño Algoritmo Memético no fue capaz de igualarse al algoritmo BVNS, probablemente debido a una convergencia prematura, aún se cree que el enfoque poblacional combinado con búsqueda local puede ser capaz de aportar buenas soluciones, por lo que se considera la posibilidad de diseñar un nuevo algoritmo con estas características.

Un área de mejora relevante para ambos algoritmos es el rediseño de la representación de las soluciones, de modo que sea posible evitar las permutaciones cíclicas equivalentes, es decir, las que se obtienen mediante el desplazamiento de todos los elementos hacia la izquierda o la derecha. Esta modificación permitiría reducir el número de soluciones visitadas y evaluadas, con lo que tendría un impacto positivo en el tiempo de ejecución.

Por otro lado, mediante algunas modificaciones en las funciones de evaluación y de vecindad, sería posible utilizar los algoritmos MA y BVNS para intentar resolver otros problemas de la familia GLP, tales como el Problema de la Anchura de Corte Cíclico [5, 46, 53] o el Anti Ancho de Banda Cíclico [34, 45].

Bibliografía

[1] D. L. Adolphson and T. C. Hu. Optimal linear ordering. *SIAM Journal on Applied Mathematics*, 25(3):403–423, 1973.

[2] R. M. Aiex, M. G. C. Resende, and C. C. Ribeiro. Ttt plots: a perl program to create time-to-target plots. *Springer-Verlag*, 2006.

[3] S. N. Bhatt and F. T. Leighton. A framework for solving VLSI graph layout problems. *Journal of Computer and System Sciences*, 28(2):300–343, 1984.

[4] L. Cavique, C. Rego, and I. Themido. Subgraph ejection chains and tabu search for the crew scheduling problem. *Journal of the Operational Research Society*, 50(6):608–616, 1999.

[5] J. Chavez and R. Trapp. The cyclic cutwidth of trees. *Discrete Applied Mathematics*, 87(1):25 – 32, 1998.

[6] Y. Chen and J. Yan. A study on cyclic bandwidth sum. *Journal of Combinatorial Optimization*, 4(2-3):295–308, 2007.

[7] F. R. K. Chung. Labelings of graphs. In L. W. Beineke and R. J. Wilson, editors, *Selected Topics in Graph Theory*, volume 3, chapter 7, pages 151–168. Academic Press, 1988.

[8] A. L. Corcoran and R. L. Wainwright. Libga: A user-friendly workbench for order-based genetic algorithm research. In *Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing: states of the art and practice*, pages 111–117. ACM, 1993.

[9] L. Davis. *Handbook of genetic algorithms*. 1991.

[10] A. Duarte, R. Martí, M. G. C. Resende, and R. M. A. Silva. GRASP with Path Relinking Heuristics for the Antibandwidth Problem. *Networks*, 58(3):171–189, 2011.

[11] A. E. Eiben and J. E. Smith. *Introduction to evolutionary computing*. Springer Science & Business Media, 2003.

[12] T. A. Feo, M. G. Resende, and S. H. Smith. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42(5):860–878, 1994.

[13] R. A. Fisher and F. Yates. *Statistical tables for biological, agricultural and medical research*. Longman, 1938.

[14] M. R. Garey and D. S. Johnson. *Computers and Intractability: A guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[15] F. Glover. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, 65(1-3):223–253, 1996.

[16] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Inc., 1989.

[17] D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. *Foundations of genetic algorithms*, 1:69–93, 1991.

[18] R. Hamon, P. Borgnat, P. Flandrin, and C. Robardet. Heuristic for solving cyclic bandwidth sum problem by following the structure of the graph. *arXiv preprint arXiv:1410.6108*, 2014.

[19] R. Hamon, P. Borgnat, P. Flandrin, and C. Robardet. Relabelling vertices according to the network structure by minimizing the cyclic bandwidth sum. *Journal of Complex Networks*, 2016.

[20] L. Harper. Optimal assignment of numbers to vertices. *Journal of SIAM*, 12(1):131–135, 1964.

[21] J. H. Holland. Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence. 1975.

[22] H. Hoos and T. Stützle. On the empirical evaluation of las vegas algorithms-position paper. Technical report, Citeseer, 1998.

[23] H. H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, 2004.

[24] H. H. Hoos and T. Stützle. Evaluating las vegas algorithms: pitfalls and remedies. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 238–245. Morgan Kaufmann Publishers Inc., 1998.

[25] H. H. Hoos and T. Stützle. *Some Surprising Regularities in the Behaviour of Stochastic Local Search*, pages 470–470. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[26] H. H. Hoos and T. Stützle. Towards a characterisation of the behaviour of stochastic local search algorithms for sat. *Artificial Intelligence*, 112(1):213–232, 1999.

[27] P. Jaccard. *Étude comparative de la distribution florale dans une portion des Alpes et du Jura*, chapter 37, pages 547–579. Bulletin de la Société Vaudoise des Sciences Naturelles, 1901.

[28] H. Jianxiu. Cyclic bandwidth sum of graphs. *Applied Mathematics-A Journal of Chinese Universities*, 16(2):115–121, 2001.

[29] Y. Jinjiang. Cyclic arrangement of graphs. In *Graph Theory Notes of New York*, pages 6–10. New York Academy of Sciences, 1995.

[30] N. Krasnogor. Memetic algorithms. In G. Rozenberg, T. Bäck, and J. N. Kok, editors, *Handbook of Natural Computing*, chapter 29, pages 905–935. Springer, 2012.

[31] Y. L. Lai and K. Williams. A survey of solved problems and applications on bandwidth, edgesum, and profile of graphs. *Graph Theory*, 31:75–94, 1999.

[32] J. K. Lenstra. *Local search in combinatorial optimization*. Princeton University Press, 1997.

[33] J. Leung, O. Vornberger, and J. Witthoff. On some variants of the bandwidth minimization problem. *SIAM Journal on Computing*, 13(3):650–667, 1984.

[34] J. Y.-T. Leung, O. Vornberger, and J. D. Witthoff. On some variants of the bandwidth minimization problem. *SIAM Journal on Computing*, 13(3):650–667, 1984.

[35] V. Liberatore. Multicast scheduling for list requests. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 1129–1137 vol.2. IEEE, 2002.

[36] Y. Lin. The cyclic bandwith problem. *Journal of Systems Science and Complexity*, 7(3):282, 1994.

[37] R. Livesley. The analysis of large structural systems. *Computer Journal*, 3(1):34–39, 1960.

[38] M. López-Ibánez, J. Dubois-Lacoste, T. Stützle, and M. Birattari. The irace package, iterated race for automatic algorithm configuration. Technical report, Citeseer, 2011.

[39] M. Lozano, A. Duarte, F. Gortázar, and R. Martí. Variable neighborhood search with ejection chains for the antibandwidth problem. *Springer Science+Business Media*, 2012.

[40] N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

[41] B. Monien and I. H. Sudborough. Embedding one interconnection network in another. *Computational Graph Theory, Computing Supplementum*, 7:257–282, 1990.

[42] P. Moscato et al. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826:1989, 1989.

[43] F. Neri and C. Cotta. Memetic algorithms and memetic computing optimization: A literature review. *Swarm and Evolutionary Computation*, 2(1):1–14, 2012.

[44] C. H. Papadimitriou. The NP-completeness of the bandwidth minimization problem. *Computing*, 16(3):263–270, 1976.

[45] A. Raspaud, O. Delmas, O. Sýkora, L. Torok, and I. Vrt'o. The cyclic antibandwidth problem. *Electronic Notes in Discrete Mathematics*, 22:223 – 227, 2005.

[46] A. Raspaud, O. Sỳkora, and I. Vrto. Congestion and dilation, similarities and differences: A survey. In *Proceedings 7th International Colloquium on Structural Information and Communication Complexity. Carleton Scientific,*, pages 269–280, 2000.

[47] E. Rodriguez-Tello, J. K. Hao, and J. Torres-Jimenez. An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem. *Computers & Operations Research*, 35(10):3331–3346, 2008.

[48] E. Rodriguez-Tello, J. K. Hao, and J. Torres-Jimenez. An improved simulated annealing algorithm for bandwidth minimization. *European Journal of Operational Research*, 185(3):1319–1335, 2008.

[49] E. Rodriguez-Tello, H. Romero-Monsivais, G. Ramirez-Torres, and F. Lardeux. Tabu search for the cyclic bandwidth problem. *Computers & Operations Research*, 57:17–32, 2015.

[50] K. Sastry, D. E. Goldberg, and G. Kendall. Genetic algorithms. In *Search methodologies*, pages 93–117. Springer, 2014.

[51] D. Satsangi. *Design and development of metaheuristic techniques for some graph layout problems*. PhD thesis, Deparment of Mathematics, Dayalbagh Educational Institute, Deemed University, May 2013.

[52] D. Satsangi, K. Srivastava, and Gursaran. General variable neighbourhood search for cyclic bandwidth sum minimization problem. In *Proceedings of the Students Conference on Engineering and Systems*, pages 1–6. IEEE Press, March 2012.

[53] H. Schröder, O. Sýykoa, and I. Vrt'o. *Cyclic Cutwidth of the Mesh*, volume 1725 of *Lecture Notes in Computer Science*, pages 449–458. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[54] J. D. Ullman. *Computational Aspects of VLSI*. Computer Science Press, 1984.

[55] M. Yagiura and T. Ibaraki. Analyses on the 2 and 3-flip neighborhoods for the MAX-SAT. *Journal of Combinatorial Optimization*, 3(1):95–114, 1999.

[56] M. Yagiura and T. Ibaraki. On metaheuristic algorithms for combinatorial optimization problems. *Systems and Computers in Japan*, 32(3):33–55, 2001.

[57] L. Yixun and Y. Jinjiang. The dual bandwidth problem for graphs. *Journal of Zhengzhou University*, 35(1):1–5, Mar. 2003.

[58] J. Yuan. Cyclic arrangement of graphs. *Graph Theory Notes of New York, New York Academy of Sciences*, pages 6–10, 1995.