TESIS DEFENDIDA POR

Hector Zatarain Aceves

Y APROBADA POR EL SIGUIENTE COMITÉ

Dr. José Alberto Hernández Zepeda
Codirector del Comité

Dr. Carlos Alberto Brizuela Rodríguez
Codirector del Comité

Dr. Israel Marck Martínez Pérez
Miembro del Comité

Dr. Joaquín Álvarez Gallegos
Miembro del Comité

Dr. Hugo Homero Hidalgo Silva
Coordinador del programa de posgrado en Ciencias de la Computación

Dr. David Hilario Covarrubias Rosales
Director de Estudios de Posgrado

5 de Diciembre de 2011

CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN SUPERIOR DE ENSENADA

PROGRAMA DE POSGRADO EN CIENCIAS

EN CIENCIAS DE LA COMPUTACIÓN

RESOLVIENDO EL PROBLEMA DE ASIGNACIÓN DE GUARDAESPALDAS UTILIZANDO ALGORITMOS GENÉTICOS

TESIS

que para cubrir parcialmente los requisitos necesarios para obtener el grado de

MAESTRO EN CIENCIAS

Presenta:

HECTOR ZATARAIN ACEVES

Ensenada, Baja California, México, Diciembre de 2011

RESUMEN de la tesis de **HECTOR ZATARAIN ACEVES**, presentada como requisito parcial para la obtención del grado de MAESTRO EN CIENCIAS en CIENCIAS DE LA COMPUTACIÓN . Ensenada, Baja California, Diciembre de 2011.

RESOLVIENDO EL PROBLEMA DE ASIGNACIÓN DE GUARDAESPALDAS UTILIZANDO ALGORITMOS GENÉTICOS

Resumen aprobado por:

Dr. José Alberto Fernández Zepeda
Codirector de Tesis

Dr. Carlos Alberto Brizuela Rodríguez
Codirector de Tesis

La cooperación entre distintos procesadores en un sistema distribuido permite cumplir un objetivo global. En sistemas distribuidos grandes, los procesadores pueden tener un objetivo individual, por lo cual su comportamiento puede ser egoísta al intentar lograr dicho objetivo. La decisión de un procesador de mejorar su función de utilidad puede entrar en conflicto con el cumplimiento del objetivo global del sistema, especialmente cuando uno o más procesadores buscan de forma egoísta incrementar unilateralmente su propia utilidad. En este documento se presenta un problema que se puede ver como un juego en un sistema distribuido. Se supone que existen dos clases de procesadores y cada clase tiene un objetivo individual opuesto entre ellos, pero todos los procesadores en el sistema deben cooperar para alcanzar un objetivo común global. En esta tesis se estudia esta problemática, siendo la distancia entre procesos el parámetro en conflicto en el sistema distribuido, y se le llama "Problema de Asignación de Guardaespaldas". Existe un algoritmo determinístico que genera soluciones aproximadas a este problema. Los algoritmos genéticos tienen muy buenos resultados para una gran variedad de problemas en donde no se conocen algoritmos determinísticos exactos y eficientes, debido a que el espacio de soluciones es muy extenso y por lo tanto, realizar una búsqueda exhaustiva determinística genera tiempos de ejecución muy grandes. En esta tesis se propone el uso de algoritmos genéticos para resolver el Problema de Asignación de Guardaespaldas. Se determina la configuración del algoritmo genético y la representación de soluciones con las que se obtiene el valor más alto al evaluar la función objetivo. Además, se proponen distintas variantes de algoritmos híbridos que combinan el algoritmo determinístico existente con el algoritmo genético propuesto. Se realizan comparaciones entre el algoritmo híbrido, el algoritmo genético y el algoritmo determinístico existente ante una gran variedad de casos de prueba. Se concluye que el algoritmo híbrido es el que obtiene mejores resultados ante cualquier tipo de caso de prueba, no obstante el algoritmo determinístico necesita un menor tiempo de ejecución.

Palabras Clave: Cómputo evolutivo, sistemas distribuidos, algoritmos genéticos, modelos de grafos aleatorios.

ABSTRACT of the thesis presented by **HECTOR ZATARAIN ACEVES**, in partial fulfillment of the requirements of the MASTER IN SCIENCES degree in COMPUTER SCIENCE . Ensenada, Baja California, December 2011.

SOLVING THE BODYGUARD ALLOCATION PROBLEM USING GENETIC ALGORITHMS TECHNIQUES

Cooperation between different processors in a distributed system allows the fulfillment of a global objective. Distributed algorithms that provide the required coordination make possible such cooperation; however, in large distributed systems, processors can also have an individual objective, so their behavior can be selfish when they attempt to improve their individual objective. The decision of a processor for improving its individual utility function may conflict with the global objective of the system, especially when one or more processors increase their individual utility in a selfish way. This document studies a computational problem that can be viewed as a game in a distributed system. It assumes the existence of two types of processors and each type has an individual objective that is opposite to the objective of the other. Also, all the processors in the system have to cooperate to achieve a common global objective. In this thesis we study this issue considering that the parameter in conflict is the distance between processors in the distributed system. This problem is the "bodyguard allocation problem". There exists a deterministic algorithm that generates approximate solutions to this problem.

Genetic algorithms have been successfully used for a wide variety of problems where there are no exact deterministic algorithms. In these problems, the searching space is very large and performing an exhaustive search is not feasible. This thesis proposes the use of genetic algorithms to generate solutions for the bodyguard allocation problem. The work presents an experimental study comparing different representations of solutions to the problem for an implementation with genetic algorithms. We determined the best configuration and the best representation for this problem. In addition to the genetic algorithm, we proposed different variants of hybrid algorithms that combine the existing deterministic algorithm with our proposed genetic algorithm. We made comparisons among the proposed genetic algorithm, the proposed hybrid algorithms and the existing deterministic algorithm, with a variety of instances, to determine the best algorithm.

The hybrid algorithm generates the highest global utility; however, the existing deterministic algorithm is faster than the hybrid and than all other genetic variants.

Keywords: Evolutionary computing, distributed systems, genetic algorithms, random graph models.

Dedicatoria

A mis padres, Héctor y Cecilia.

A mi esposa, Karina.

Agradecimientos

A Karina por alentarme a emprender esta aventura juntos y darme ánimo cuando más lo necesitaba. Gracias por tu cariño, por cocinar tan rico, pero sobretodo por ser parte de mi vida.

A mis padres por su cariño y apoyo incondicional, gracias por ser mi gran orgullo y ejemplo a seguir.

A mis codirectores, el Dr. Carlos Alberto Brizuela y el Dr. José Alberto Fernández, que me guiaron durante la elaboración de mi tesis.

A mi comité de tesis, por sus observaciones y críticas constructivas para mejorar mi investigación.

A mis compañeros de la generación 2009, por su compañía durante las largas horas de estudio.

A todos mis compañeros del cubo 103, por hacer la convivencia de cada día más amena.

A todos los investigadores del posgrado en ciencias de la computación por su enseñanza académica.

Al personal del departamento de ciencias de la computación por hacer mi estancia en la institución lo más agradable posible.

Al Centro de Investigación Científica y de Educación Superior de Ensenada.

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por brindarme el apoyo económico para realizar esta maestría.

Contenido

Contenido (continuación)

Contenido (continuación)

Lista de Figuras

Lista de Figuras (continuación)

Lista de Tablas

Lista de Tablas (continuación)

Lista de Tablas (continuación)

Lista de Algoritmos

Capítulo I

Introducción

I.1. Planteamiento del problema

La cooperación entre distintos procesadores en un sistema distribuido permite cumplir un objetivo global. Los algoritmos distribuidos que proveen la coordinación requerida hacen posible tal cooperación; sin embargo, en sistemas distribuidos grandes como la Internet, los procesadores pueden tener un objetivo individual, por lo cual su comportamiento puede ser egoísta al intentar mejorar este objetivo. Por ejemplo, si se considera una red que utilice el protocolo *Peer-to-Peer* (P2P) (Schollmeier, 2001), la cual tiene el propósito de compartir archivos entre las entidades de la red (como objetivo global), si cada entidad se comporta de forma egoísta y sólo descarga archivos sin compartir nada, se fracasa en el objetivo global ya que la cooperación es fundamental en este tipo de redes. Cualquier objetivo individual se puede representar como una función objetivo a ser maximizada o minimizada. Si se habla de maximizar, se puede ver la función objetivo como una función de utilidad que represente el beneficio del procesador. La decisión de un procesador de mejorar su función de utilidad individual puede estar en conflicto con el cumplimiento del objetivo global del sistema, especialmente cuando uno o más procesadores buscan de forma egoísta incrementar su utilidad individual. El uso de objetivos individuales en sistemas distribuidos genera conflictos porque cada procesador intenta beneficiarse a sí mismo tanto como le sea posible y al mismo tiempo, cada procesador intenta cooperar para alcanzar el objetivo global. Los sistemas que se comportan de

esta forma se han estudiado de forma extensa en la teoría de juegos (Neumann *et al.*, 2007). La teoría de juegos estudia situaciones de competencia y cooperación entre individuos en donde las decisiones que toman dependen de la interacción entre ellos. Los sistemas distribuidos proveen un servicio computacional como su objetivo global. Se dice que los sistemas fallan cuando su objetivo global no se cumple. En este contexto, la cooperación entre procesos es más valiosa que el individualismo. Es deseable que los procesos primero cooperen entre ellos hasta alcanzar la meta global, y después de esto mejoren su objetivo individual; sin embargo, este enfoque es complicado de lograr dado que a veces la utilidad individual de un procesador egoísta puede ser más alta que la utilidad que se obtiene al cooperar con otros procesos (Dasgupta *et al.*, 2006). Un sistema distribuido efectivo debe ser capaz de distinguir entre estos objetivos y elegir el objetivo global sobre el individual. En este documento se presenta un problema que se puede ver como un juego en un sistema distribuido. Se supone que existen dos clases de procesadores y cada clase tiene un objetivo individual opuesto entre ellos, pero todos los procesadores en el sistema deben cooperar para alcanzar un objetivo común global. La función de utilidad del procesador depende de ciertos parámetros en el sistema, tal que los de una clase desean maximizar el valor del parámetro mientras que la otra clase de procesadores desean minimizarlo. Cada procesador incrementa su utilidad individual basado en la maximización o minimización de este parámetro y al mismo tiempo, contribuye a la utilidad global del sistema. En esta tesis se estudia esta problemática en donde el parámetro a maximizar/minimizar es la distancia entre procesadores en el sistema distribuido, y se le llama el "Problema de Asignación de Guardaespaldas" (PAG). El PAG se describe de la siguiente forma: Existe una persona (cliente) a quien protegen $n-1$ guardaespaldas alojados en una casa. La casa se puede representar como un grafo, donde cada vértice representa un cuarto y cada arista representa un pasillo

que conecta dos cuartos. Todos los guardaespaldas y el cliente se localizan en diferentes cuartos. El objetivo de cualquier guardaespaldas es el de proteger al cliente de la mejor forma posible. Cada guardaespaldas puede proteger solo un pasillo. Los pasillos que nadie protege se cierran. Cada pasillo abierto permite la comunicación entre los respectivos cuartos. Se busca que todos los guardaespaldas y el cliente estén comunicados (esta restricción hace que los cuartos y pasillos abiertos se vean como un árbol de esparcimiento del grafo original). El criterio usado para cerrar los pasillos se basa en la habilidad individual de cada guardaespaldas. Se requiere que los guardaespaldas expertos estén lo más cerca posible del cliente, mientras que los novatos estén lo más lejanos posibles del cliente. Entonces, los expertos quieren minimizar su distancia con respecto al cliente y los novatos quieren maximizarla. Se puede modelar el PAG con un sistema distribuido en donde la configuración de la red es análoga a una entrada arbitraria del problema. El problema seria encontrar la asignación de los guardaespaldas a los cuartos de tal forma que se obtenga la mejor la protección del cliente, es decir, el árbol de esparcimiento que maximize la seguridad del cliente.

I.2. Objetivos

En esta sección se definen los dos objetivos generales de la investigación así como una serie de objetivos particulares que ayudan a cumplir con ellos.

I.2.1. Objetivos generales

Los principales objetivos de la investigación, se basan en responder al siguiente par de preguntas:

¿Cómo es el desempeño de los algoritmos evolutivos para resolver el PAG en com-

paración con algoritmos determinísticos?

¿Cuál es la técnica evolutiva que tiene mejores resultados para resolver el PAG?

I.2.2. Objetivos específicos

Para lograr cumplir los objetivos generales de la investigación, es necesario lograr los siguientes objetivos particulares.

1. Analizar distintas representaciones de genotipos y su efecto en el desempeño de los algoritmos evolutivos.

2. Diseñar un algoritmo genético para el PAG que encuentre soluciones mejores que la versión determinística existente.

3. Adecuar los parámetros de probabilidad de mutación y cruzamiento de los algoritmos evolutivos de tal forma que se obtengan las mejores soluciones posibles.

4. Definir formalmente los tipos de operadores de mutación, cruzamiento y selección de los algoritmos evolutivos.

5. Establecer un mecanismo que estime el grado de calidad de una solución dado una solución al PAG.

I.2.3. Preguntas de investigación

1. ¿Cuál es la mejor representación de un genotipo de los algoritmos genéticos que resuelven el PAG?

2. ¿Qué efectos tienen los ajustes de probabilidad de cruzamiento y mutación en los resultados de los algoritmos evolutivos?

3. ¿Cuáles son los operadores evolutivos que mejor se ajustan a los algoritmos genéticos para el PAG?

4. ¿Cómo se comportan los algoritmos evolutivos ante las variantes del PAG?

I.3. Motivación

Se ha demostrado de forma empírica que los algoritmos evolutivos tienen muy buenos resultados para una gran variedad de problemas en donde no existen algoritmos determinísticos exactos (Gen y Cheng, 1997), (Wang y Kusiak, 2001); lo anterior debido a que el espacio de soluciones es muy extenso y por lo tanto, realizar una búsqueda exhaustiva determinística genera tiempos de ejecución muy grandes. En Fajardo y Fernández (2010) se propone un algoritmo determinístico para solucionar el PAG; este algoritmo no siempre encuentra el óptimo global debido a que se queda estancado en óptimos locales. Una característica interesante de los algoritmos evolutivos es que por su naturaleza, permiten explorar de forma amplia el espacio de soluciones; para realizarlo, utilizan operadores de variación que, si se implementan de forma adecuada, logran evitar el estancamiento en óptimos locales.

El problema de asignación de guardaespaldas se puede plantear como un problema en el área de diseño de redes. Una aplicación puede ser la de permitir que una red sea capaz de reconfigurarse con base en sus necesidades de transmisión a un servidor fijo, de tal forma que las aristas que se elijan sean las que provean mayor velocidad de transmisión de datos para los vértices; sin embargo, algunos vértices pueden tener mayor prioridad de transmisión que otros y además se puede considerar que la red puede cambiar constantemente, agregando o eliminando vértices. Por lo anterior, es necesario que la red sea capaz de reconfigurarse para sobrellevar estos cambios manteniendo a

la vez una buena velocidad de transmisión. En otros problemas del área de sistemas distribuidos, el PAG puede ser una buena herramienta para asignar los recursos de la red. Por ejemplo, si se cuenta con un sistema de respaldo espejo en donde se replican todos los datos de un servidor a otro, se desea que el servidor espejo se encuentre lo más alejado posible del servidor original, para que en el caso de que ocurra alguna contingencia en el servidor original, la pérdida de datos se minimice. Otro ejemplo puede darse, en una red de sensores inalámbricos alimentados por baterías; los sensores tienen que estar en constante comunicación con un servidor por medio de mensajes, para llevar a cabo esta comunicación se genera un árbol para conocer la ruta al servidor. Cada sensor reenvía los mensajes que recibe de otros sensores hasta que se entreguen al servidor. Cada mensaje que un sensor envía, descarga su batería, por lo que los sensores que estan más cerca del servidor se descargan más rapido que los que están más alejados. Es por ello que se tiene que encontrar el árbol para comunicar los sensores tomando en cuenta la carga disponible de cada sensor y a su vez tener la capacidad de reconfigurar este árbol para aumentar la vida útil del mismo.

I.4. Hipótesis y contribución

La principal hipótesis que se plantea para este trabajo de investigación es la siguiente: El uso de algoritmos genéticos para resolver el PAG permite obtener mejores resultados al evaluar la función objetivo que el algoritmo determinístico propuesto por Fajardo y Fernández (2010).

Las principales contribuciones de esta investigación se mencionan a continuación:

1. Se hace un análisis para determinar los mejores operadores de variación de distintas representaciones para algoritmos genéticos.

2. Se implementan varios algoritmos genéticos para generar soluciones a un conflicto entre procesos de un sistema distribuido.

3. Se evalúa el desempeño de varias representaciones para el PAG utilizando algoritmos genéticos.

4. Se mejoran los resultados de un algoritmo determinístico conocido para el PAG utilizando algoritmos genéticos.

I.5. Organización de tesis

Este documento de tesis está organizado de la siguiente manera: en el Capítulo I se da una introducción al problema, así como la motivación y objetivos de la investigación. En el Capítulo II se definen los términos más relevantes para realizar esta investigación, además de describir y analizar las distintas representaciones de árboles para algoritmos genéticos. En el Capítulo III se define de manera formal el problema de asignación de guardaespaldas, así como sus antecedentes. En el Capítulo IV se explican y comparan los métodos para generar entradas válidas para el algoritmo genético. En el Capítulo V se describe el funcionamiento del algoritmo genético y sus variantes. En el Capítulo VI se realizan pruebas preliminares por cada representación para determinar con qué operadores se obtienen mejores resultados. En el Capítulo VII se hacen comparaciones entre las distintas representaciones, las variantes de algoritmos genéticos y el algoritmo genético que genera mejores resultados con el algoritmo determinístico propuesto por Fajardo y Fernández (2010). En el Capítulo VIII se presentan las conclusiones de la investigación y se mencionan algunas alternativas para trabajos futuros relacionados con el problemas de asignación de guardaespaldas y los algoritmos genéticos.

Capítulo II

Marco Teórico

Una heurística es una técnica para resolver problemas con base en la experiencia, utilizando información para controlar el manejo de la resolución del problema. El presente proyecto de investigación se enfoca en resolver el problema de asignación de guardaespaldas por medio de técnicas de algoritmos evolutivos, así como algunas heurísticas de la teoría de juegos. A continuación se da una breve descripción de estos conceptos.

II.1. Teoría de juegos

Con base en las definiciones de Nisan (2007), la teoría de juegos estudia situaciones de competencia y cooperación entre individuos, en donde las decisiones que ellos toman dependen de la interacción que tienen entre sí. La teoría de juegos utiliza modelos para estudiar interacciones de distintos jugadores con estructuras formalizadas de incentivos para llevar a cabo procesos de decisión. En la teoría de juegos los problemas se plantean como juegos, en donde participan dos o más jugadores, cuando estos jugadores tienen un objetivo global común, se busca llegar a un equilibrio de Nash. El equilibrio de Nash captura la noción de una solución estable, ésta se define como una solución en la que ningún jugador puede individualmente mejorar su estado actual.

II.2. Cómputo evolutivo

El cómputo evolutivo (CE) se implementa por medio de algoritmos evolutivos (AEs). La motivación de los AEs es la de imitar la evolución biológica y aprovechar el poder de adaptación. Sus características principales como se establecen en Eiben y Smith (2003) se mencionan a continuación:

- Existe una población de individuos en un ambiente con recursos limitados.
- La competencia por los recursos causa la selección de los individuos mejor adaptados a su entorno.
- Estos individuos actúan como semilla de nuevas generaciones a través de mecanismos de mutación y recombinación.
- La selección natural causa un aumento en la aptitud de la población a través del tiempo.

Seredynski (2006) menciona y describe las técnicas más reconocidas de algoritmos evolutivos: estrategias evolutivas (EE), algoritmos genéticos (AG), programación genética (PG), programación evolutiva (PE) y sistema de aprendizaje clasificador (SAC). Estas técnicas son muy similares entre sí pero difieren en algunos detalles de implementación y se eligen dependiendo la naturaleza del problema o de cómo se modele su representación. El enfoque de algoritmos genéticos (AGs) es actualmente el que más se utiliza en el cómputo evolutivo. En Seredynski (2006) se analizan más a fondo los paradigmas evolutivos y sus características.

El cómputo evolutivo es una área de investigación en la cual una componente importante de ella se orienta hacia la experimentación. A pesar del estado emergente del

campo, el CE ha probado su potencial al resolver distintos problemas teóricos y prácticos. Las técnicas de CE se han aplicado de forma satisfactoria para resolver distintos problemas comerciales como colocación de torres telefónicas, diseño de redes de fibra óptica, sistema de seguridad comercial o procesos de calendarización.

Los efectos acumulativos de los operadores de selección, cruzamiento y mutación en el proceso evolutivo se pueden estudiar modelándolos mediante cadenas de Markov. Las propiedades de los algoritmos genéticos se pueden estudiar modelando el algoritmo como un sistema dinámico o el mismo puede aproximarse también con técnicas de mecánica estadística. El concepto de paisaje (*landscape*) y algunas metodologías como la representación de Walsh pueden utilizarse para predecir el desempeño de un algoritmo genético para resolver un problema (Vose y Wright (1998), Goldberg (1989a), Goldberg (1989b)).

Mientras que la teoría de CE está aun en desarrollo, algunos avances en la construcción de esta teoría pueden ser notorios. Por ejemplo, Wolpert y Macready (1997) proponen el teorema de no almuerzo gratis (*no free lunch theorem*), el cual demuestra que el desempeño de todas las heurísticas y algoritmos de búsqueda promediados sobre todas las funciones posibles, es la misma si éstas satisfacen ciertas condiciones, es decir que cualquier par de algoritmos son equivalentes si se promedia su desempeño ante todos los problemas posibles. Los mismos autores Wolpert y Macready (2005) demuestran que puede existir el almuerzo gratis en algoritmos coevolutivos. Es decir utilizando múltiples escenarios de búsqueda y haciendo interactuar a distintos agentes para que trabajen por un objetivo en común (para el caso particular de los problemas de auto juego, *self-play*), donde los agentes trabajan juntos para producir un campeón, quien después adquiere mayor antagonismo en un juego de múltiples jugadores. Esto contrasta con la optimización tradicional donde no existe el almuerzo gratis; en el caso

de la coevolución, algunos algoritmos tienen mejor desempeño que otros algoritmos al promediar su desempeño sobre todos los problemas posibles.

El campo del cómputo evolutivo sirve además como plataforma para el desarrollo de nuevos algoritmos de búsqueda basados en población. Algunos ejemplos de algoritmos de búsqueda fuertemente ligados con el cómputo evolutivo son la evolución diferencial (Storn y Price, 1997), algoritmos meméticos (Moscato, 1999), algoritmos culturales (Reynolds, 1994) o inteligencia de enjambre (Beni y Wang, 1993). Los sistemas inmunes artificiales (Farmer *et al.*, 1986) y optimización por enjambre de partículas (Beni y Wang, 1993) son algoritmos de búsqueda que se basan en nuevos paradigmas y su intersección con conceptos evolutivos es visible.

II.3. Algoritmos genéticos

Un algoritmo genético (AG) es una heurística de búsqueda que imita el proceso de evolución natural. Esta heurística se utiliza para generar soluciones para problemas de búsqueda y optimización. Los AG pertenecen a un grupo mayor de algoritmos evolutivos que a su vez pertenecen al cómputo evolutivo. Este tipo de algoritmos se dividen en fases inspiradas por la evolución natural, tales como herencia, mutación, selección y recombinación (también llamada cruzamiento).

En un AG se utiliza una población de cadenas (genotipos), las cuales codifican soluciones candidatas (individuos) para un problema de optimización; estos individuos posteriormente evolucionan hacia mejores soluciones. La evolución usualmente empieza a partir de una población de individuos generados aleatoriamente y ocurre a través de generaciones. En cada generación, la aptitud de cada individuo de la población se evalúa, se seleccionan varios individuos de la población con base en su aptitud y se

modifican (recombinan y mutan aleatoriamente) para formar una nueva población. La nueva población se utiliza en la próxima generación del algoritmo por medio de un criterio de selección. Comúnmente, el algoritmo termina cuando un número máximo de generaciones se produjeron o se alcanzó un nivel satisfactorio de la aptitud de la población. En la Figura 1 se muestra el esquema general de un algoritmo evolutivo.

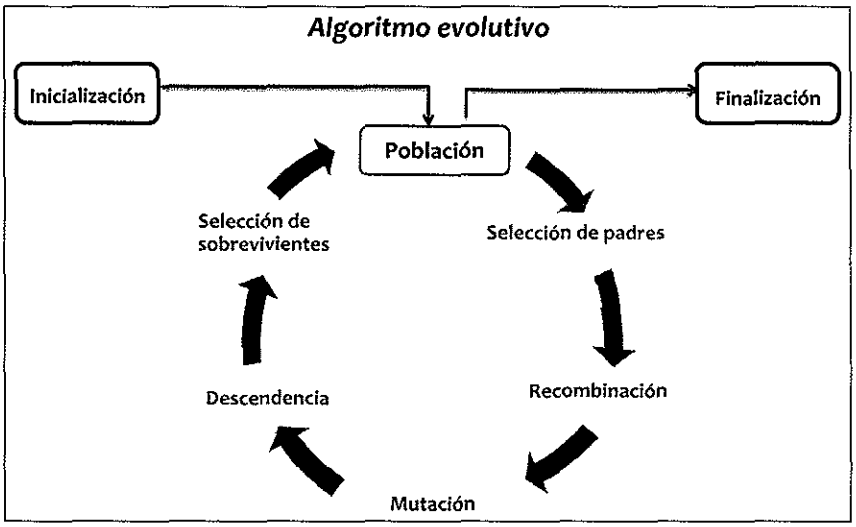


Figura 1: Esquema general de un algoritmo evolutivo.

Un algoritmo genético requiere:

1. Una representación genética del dominio de la solución.

2. Una función de aptitud para evaluar la solución.

El primer punto se desarrolla en la Sección II.4, mientras que el segundo punto se explica en la Sección III.1, en la descripción del problema.

Los AG fueron inicialmente propuestos por Holland (1992) como una forma de estudiar el comportamiento adaptativo. Holland en su definición de AG, definió una representación binaria, una selección proporcional por aptitud, una baja probabilidad de mutación y un énfasis en la recombinación inspirada genéticamente para generar

nuevas soluciones candidatas. A esta configuración se le conoce como "algoritmo genético simple" o "algoritmo genético canónico".

II.4. Representaciones

El primer paso para definir un algoritmo evolutivo es el establecer un puente entre el contexto del problema original y el espacio de solución del problema donde la evolución ocurre. Uno de los puntos clave al utilizar algoritmos evolutivos es definir la representación o codificación que tendrá una solución candidata factible al problema, ya que el desempeño de un algoritmo evolutivo depende de la interacción de la codificación de sus soluciones candidatas con los operadores de variación que se le aplican. A esta representación se le llama genotipo. El genotipo debe ser una estructura de datos que decodifique su fenotipo, es decir que sea una fiel representación de la solución en el problema real evitando redundancia. El proceso de elegir una representación correcta implica definir un genotipo y el mapeo del genotipo al fenotipo; esta elección es una de las partes más importantes para diseñar un buen algoritmo evolutivo. A continuación se describen las propiedades o características que debe tener una buena representación tanto en general como para nuestro problema en particular.

II.4.1. Características deseables en representaciones para árboles

Como una solución al PAG es un árbol de esparcimiento, en esta investigación se estudiaron las características que debe cumplir una buena representación para árboles; algunas de estas características se mencionan a continuación:

- Espacio: Los elementos de la representación o cromosomas no deben requerir grandes cantidades de memoria o estructuras de datos muy complejas.
- Tiempo: La evaluación, recombinación y mutación de los cromosomas debe tener una complejidad temporal pequeña. Cuando los cromosomas representan árboles de esparcimiento, la evaluación incluye la decodificación del cromosoma que identifica el árbol de esparcimiento que representa.
- Factibilidad: Todos los cromosomas, particularmente aquellos generados por cruzamiento y mutación deben representar soluciones factibles.
- Unicidad: Se desea que el mapeo de los cromosomas a las soluciones (decodificación) sea 1 a 1, el caso menos deseado es el mapeo 1 a n.
- Herencia persistente (*Heritability*): La descendecia de un cruce debe representar una solución que combine subestructuras de las soluciones padres.
- Localidad: Un cromosoma mutado normalmente debe representar una solución similar a la que representa el cromosoma original.
- Tendencia: En general, las representaciones deben mapear a todo el espacio de soluciones de forma similar, aunque puede ser una ventaja si favorecen a las soluciones que se encuentran cerca del óptimo.
- Restricciones: La decodificación de los cromosomas y los operadores de cruce y mutación deben ser capaces de respetar las restricciones específicas del problema.
- Híbridos: Los operadores de variación deben ser capaces de incorporar heurísticas dependientes del problema.

- Grafos dispersos: Algunas codificaciones pueden representar árboles de esparcimiento solo en grafos completos. La codificación se debe poder utilizar para representar subgrafos de grafos completos.

El PAG tiene como entrada un grafo no dirigido y su salida es un árbol de esparcimiento, por lo tanto tiene ciertas similitudes en cuanto a estructura con el problema de encontrar un árbol de esparcimiento mínimo (*Minimum Spanning Tree* o MST) en un grafo. El objetivo de ambos problemas es distinto; más sin embargo, se pueden modelar de una forma similar.

II.4.2. Representaciones para árboles de esparcimiento

Una solución válida para el PAG es un árbol de esparcimiento, por lo cual las representaciones que se abordan en esta sección son para la codificación de árboles.

Vector característico

Esta representación mapea un árbol de esparcimiento a un vector binario. Dado un grafo con n vértices y m aristas, cada una de las aristas se etiqueta con un identificador único del conjunto $\{1, \ldots, m\}$, un árbol de esparcimiento se representa de tal forma que si la arista i del grafo original se incluye en el árbol se escribe un 1 en la posición i del vector binario y un 0 en caso contrario; en el Algoritmo 1 se ilustra este procedimiento. Esta codificación toma un tiempo de $O(n^2)$ en el peor de los casos, debido a que el vector característico tiene una longitud de $m = n(n-1)/2$ en un grafo completo (requiere $O(m)$ espacio). El tamaño del espacio de búsqueda es $2^{n(n-1)/2}$, que es el número de vectores binarios distintos que se pueden obtener; sin embargo, la mayoría de estos vectores no representan soluciones factibles, ya que un grafo completo tiene solo n^{n-2} árboles

de esparcimiento. Esto presenta problemas al generar soluciones aleatorias y al aplicar los operadores de variación, ya que se pueden obtener soluciones inválidas; para evitar este problema es necesario implementar ciertas heurísticas. Una heurística puede ser que cada vez que se encuentre una solución que no satisface los requerimientos para ser una solución válida, ésta se penalice en su aptitud para tener pocas posibilidades de sobrevivir durante el ciclo evolutivo. Otra posible heurística es implementar un algoritmo de reparación de las representaciones inválidas. Esto último es costoso en tiempo computacional y decrementa la eficiencia de las representaciones que la utiliza.

Algoritmo 1: Codificación vector característico.

Input : Grafo $G = (V, E)$ y un árbol de esparcimiento $T = (V, E')$ donde $E' \subseteq E$.

Output : Vector binario B de tamaño $|E'|$

```
1 for each e ∈ E do
2     if e_i exist in T then
3         B[e_i] = 1;
4     else
5         B[e_i] = 0;
6     end
7 end
```

En la Figura 2a se observa un ejemplo de un grafo; en la Figura 2b se muestra uno de sus posibles árboles de esparcimiento; finalmente, en la Figura 2c se muestra el vector característico del árbol de esparcimiento.

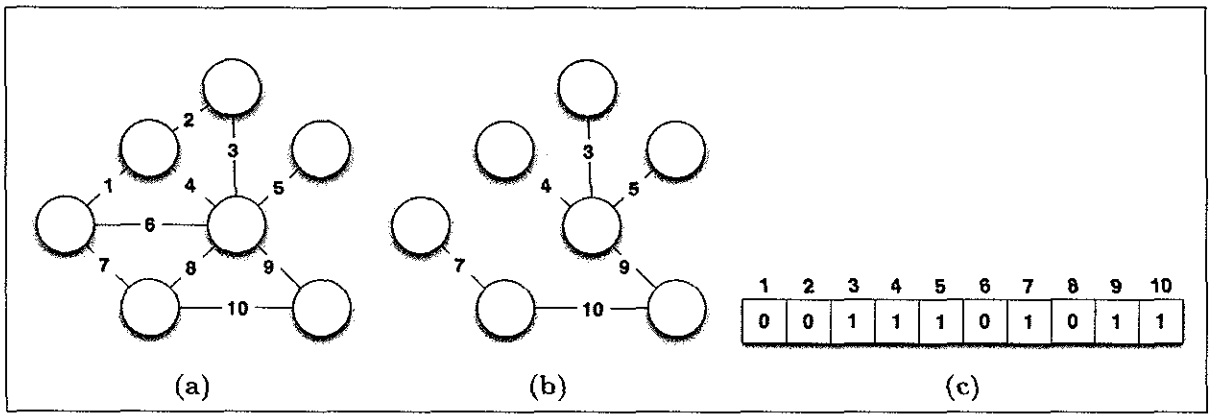


Figura 2: Ejemplo de mapeo para la representación de vector característico. a) Grafo original; b) Árbol de esparcimiento; c) Vector binario.

Vector predecesor

Esta representación funciona al designar un vértice como la raíz r del árbol y guardar los predecesores de cada vértice del árbol enraizado en r. De acuerdo a Cormen *et al.* (2001), se puede obtener un listado de predecesores (padres) en $O(h)$ para árboles binarios y $O(\log n)$ para árboles no binarios, donde h es la altura del árbol.

Palmer y Kershenbaum (1995) definen $Pred[i] = j$, donde j es el primer vértice en la trayectoria de i a r (vértice raíz) en T; para el caso de r, se dice que $Pred[r] = r$. Por lo anterior, el vector predecesor es un vector numérico de tamaño n, aunque en algunos trabajos (Raidl y Julstrom, 2003) el predecesor del vértice raíz se omite (apunta a nulo), dando como resultado un vector de tamaño $n - 1$; sin embargo, para ambos casos el tiempo de codificación es el mismo $O(n)$.

Para el proceso del mapeo de un árbol a un vector predecesor se utiliza la función $neighborhood(v) = \{u|(u, v) \in E'\}$ que obtiene todos los vecinos de v en el árbol. El Algoritmo 2 utiliza el Algoritmo 3 (que es recursivo) para codificar el vector predecesor; sin embargo, este proceso se puede implementar de forma iterativa si se utiliza una pila para almacenar los vértices por visitar. El mapeo funciona de la siguiente manera: dado

un árbol con n vértices etiquetados con identificadores únicos, se crea un vector de tamaño n, en donde se van almacenando los predecesores o padres de cada vértice del árbol empezando por el vértice raíz hasta las hojas del árbol. Este proceso también se puede realizar en un grafo utilizando una búsqueda por anchura (Cormen *et al.*, 2001), la cual toma un tiempo de $O(V + E)$.

En la Figura 3a se muestra un árbol de esparcimiento enraizado en el vértice 4 y su vector de predecesores se puede observar en la Figura 3b. Cabe señalar que esta codificación puede generar soluciones no válidas al problema, por lo que se tiene que verificar si la solución generada es válida y utilizar penalización o reparación de las soluciones inválidas, lo cual es costoso en tiempo.

Algoritmo 2: Codificación vector predecesor.

Input : Árbol de esparcimiento $T = (V, E')$.

Output : Vector numérico de tamaño $|V| - 1$

```
1 v = root;
2 Pred[v] = v;
3 setPredecessors(T, v);
```

Algoritmo 3: Asigna predecesores (setPredecessors).

Input : Árbol de esparcimiento $T = (V, E')$, vértice padre v

```
1 for each u ∈ neighborhood(v) do
2     if Pred[u] == null and u! = root then
3         Pred[u] = v;
4         setPredecessors(T, u);
5     end
6 end
```

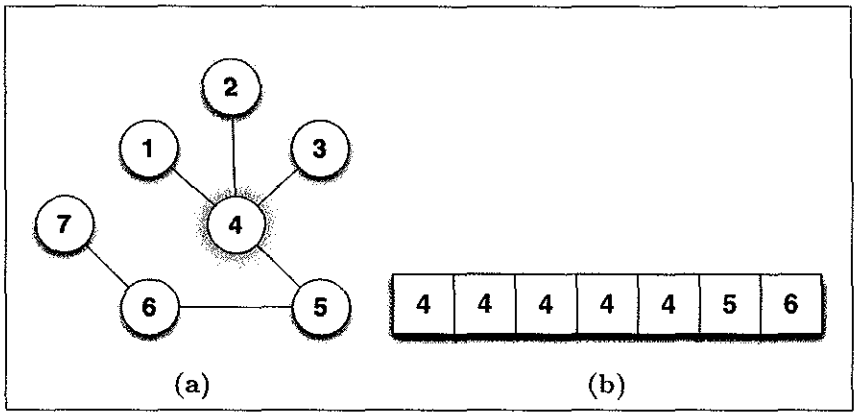


Figura 3: Ejemplo de mapeo para la representación de vector predecesor.
a) Árbol enraizado en el vértice 4; b) Vector de predecesores.

Vector de Prüfer

La formula de Cayley (1889) identifica n^{n-2} árboles de esparcimiento en un grafo completo con n vértices. Prüfer (1918) presentó una demostración de este resultado, dando un mapeo entre árboles de esparcimiento con n vértices y vectores de longitud $n-2$ con los vértices etiquetados en forma numérica (a estos se les llaman vectores o números de Prüfer). Esta representación tiene la propiedad de que se puede codificar y decodificar en $O(n \log n)$, además de que un vector de Prüfer representa un árbol único y viceversa, es decir el mapeo es biyectivo.

Esta representación se utilizó mucho hasta que se comprobó que tenía muy mala capacidad de herencia persistente (*heritability*) y localidad en algoritmos evolutivos. Esto lo demuestran Gottlieb *et al.* (2001) por medio de una investigación empírica en cuatro problemas NP-Difícil que involucran árboles de esparcimiento.

Una característica notable en esta representación es que el grado de cada vértice se identifica fácilmente a partir del vector de Prüfer, ya que cada grado del vértice es uno más que el número de veces que aparece el identificador del vértice en el vector de Prüfer. Esta característica la hace una representación útil para problemas en los cuales

la restricción es el grado de los vértices (Knowles y Corne, 2000).

El Algoritmo 4 muestra la codificación del vector de Prüfer. El algoritmo utiliza la función $degree(v)$, que regresa el grado del vértice v, y la función min, que regresa el vértice con la etiqueta menor de un conjunto dado. Para la implementación del vector de Prüfer se utiliza la mutación aleatoria gen por gen y cruzamiento uniforme de acuerdo a Knowles y Corne (2001a). Un ejemplo del mapeo para la codificación de Prüfer se ilustra en la Figura 4.

Algoritmo 4: Codificación vector Prüfer.

Input : Árbol de esparcimiento $T = (V, E)$.

Output : Vector de Prüfer P de tamaño $|V| - 2$

1 $n = |V| - 2$;

2 **for** $con = 0$ *to* n **do**

▷ Sea i la hoja con la etiqueta más pequeña en T

3 $i = min\{v \mid degree(v) = 1\}$;

▷ Se inserta el predecesor de i al vector de Prüfer

4 $P[con] = Pred[i]$;

▷ Se elimina el vértice i y la arista (i, j)

5 $V = V - i$;

6 $E = E - (i, j)$;

7 **end**

▷ Se detiene cuando quedan dos vértices en el árbol

8 *return* P

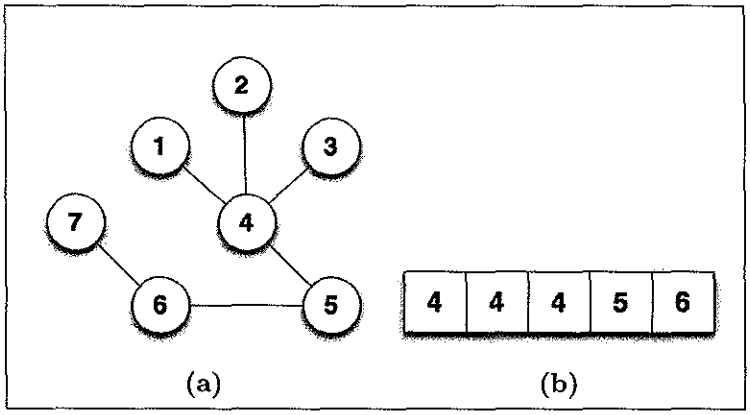


Figura 4: Ejemplo de mapeo para la representación de vector Prüfer.
a) Árbol de ejemplo; b) Vector de Prüfer.

Código blob

El código blob (*blob code*) se basa en el número de Prüfer y al igual que éste, el algoritmo requiere un espacio de $n-2$; se supone que los vértices están numerados de 0 a $n-1$. En el mapeo original de Picciotto (1999) se utiliza el concepto de supervértice, el cual representa un subconjunto de vértices; a este supervértice también se le llama *blob* y en el algoritmo se le trata como un solo vértice. El funcionamiento de la codificación es el siguiente: dado un árbol enraizado en el vértice 0, el algoritmo va tomando los vértices en orden decreciente, cada vértice se separa de su padre y se agrega al supervértice *blob*; de esta forma, este supervértice solamente tiene un padre en el árbol (un vértice normal) pero cada vértice dentro del *blob* conserva su propio subárbol. Algunos vértices forzan al *blob* a cambiar de padre, otros no. La codificación utiliza la función $p(v)$, que regresa el padre del vértice v, y la función $path(v) \bigcap blob$, que regresa *true* si la trayectoria del vértice v al vértice raíz pasa por el *blob* y $false$ o $\emptyset$ de lo contrario.

Los algoritmos 5 y 6 muestran el proceso de codificación y decodificación para el blob code, respectivamente, y se basan en el mapeo original propuesto por Picciotto (1999). Ambos algoritmos tienen un tiempo de ejecución de $O(n^2)$. Un ejemplo del proceso de

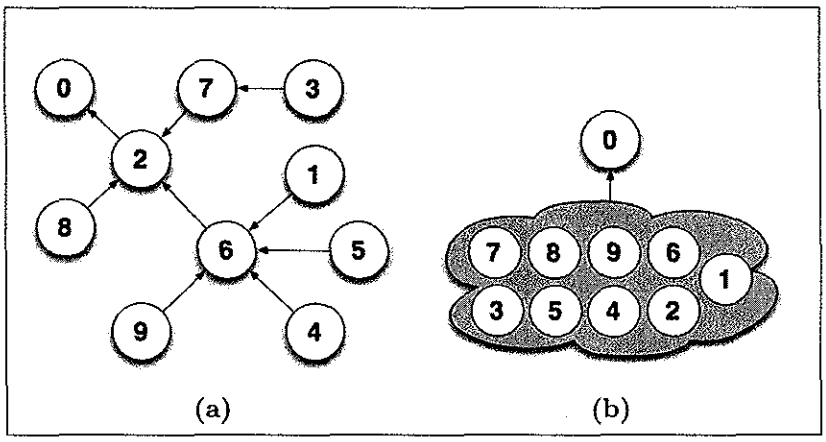


Figura 5: Árboles de ejemplo para la codificación Blob.
a) Árbol inicial; b) Árbol final.

codificación de este algoritmo se puede observar en la Figura 5, donde se muestra el árbol inicial y el árbol final al utilizar el código blob. En la Figura 6 se muestra el estado del árbol en la primera iteración, además de las variables, las operaciones y el resultado de la comparación de la línea 6 del Algoritmo 5; en la Figura 7 se muestran también las variables y las operaciones de la última iteración, en este par de figuras se aprecian las dos diferentes acciones que se toman con base en la comparación de la línea 6 del Algoritmo 5. Julstrom (2001) compara Prüfer con el código blob siendo este último el que obtiene los mejores resultados ante el problema One Max Tree. Paulden y Smith (2006b) mejoraron el tiempo de ejecución a $O(n)$. Caminiti y Petreschi (2010) diseñaron un algoritmo paralelo del código blob en una EREW PRAM, el cual toma un tiempo de $O(\log n)$ con $O(n/\log n)$ operaciones; para la decodificación fue necesaria una CREW con tiempo de $O(\log n)$ con $O(n)$ operaciones.

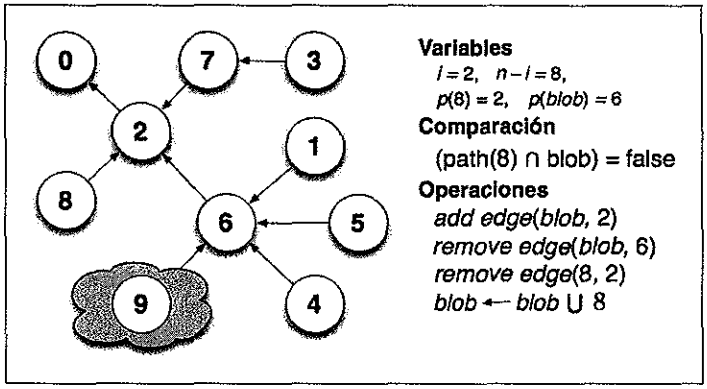


Figura 6: Árbol, variables y operaciones en la primera iteración del Algoritmo 5 para el código Blob.

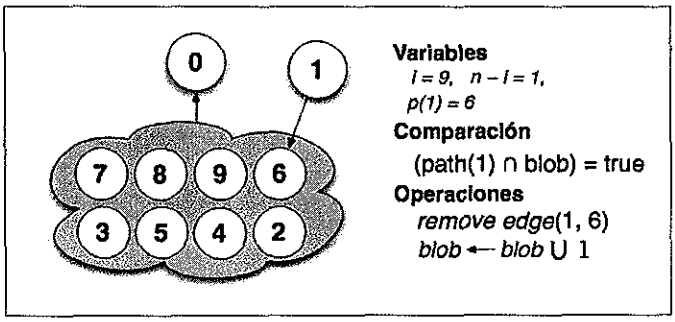


Figura 7: Árbol, variables y operaciones en la última iteración del Algoritmo 5 para el código Blob.

Algoritmo 5: Codificación código blob de Picciotto.

Input : Árbol de esparcimiento $T = (V, E')$.

Output : code: Código blob de tamaño $|V| - 2$

```
1  code = ∅;
2  T ← add edge(blob, p(n));
3  T ← remove edge(n, p(n));
4  blob = blob ∪ {n};
5  for i = 1 to n do
       ▷ Si la trayectoria del vértice n − i a la raíz pasa por el blob
6      if path(n − i) ⋂ blob ≠ ∅ then
7          code[n − i] = p(n − i);
8      else
9          T ← add edge(blob, p(n − i));
10         T ← remove edge(blob, p(blob));
11         code[n − i] = p(blob);
12     end
13     blob = blob ∪ n − i;
14     T ← remove edge(n − i, p(n − i));
15 end
16 return code
```

Algoritmo 6: Decodificación código blob de Piccioto.

Input : code: Código blob de tamaño $|V| - 2$, Grafo $G = (V, E)$

Output : Árbol de esparcimiento $T = (V, E)$.

```
1  T ← add edge(blob, 0);
2  blob = {1, 2, ...n − 1};
3  for i = 1 to n − 2 do
       ▷ Si la trayectoria del vértice n − i a la raíz pasa por el blob
4      blob = blob − i;
5      if path(a_i) then
6          T ← add edge(i, a_i);
7      else
8          T ← add edge(i, p(blob));
9          T ← add edge(blob, a_i);
10         T ← remove edge(blob, p(blob));
11     end
12     blob ← (n − 1) in all edges;
13 end
14 return T
```

Codificación con prioridades en aristas y vértices

Cuando se tiene una red con algunas aristas prioritarias o de alta calidad y se desea que la solución generada contenga estas aristas. En los algoritmos evolutivos los descendientes no necesariamente heredan las aristas buenas o de alta calidad de sus padres, por lo regular solo tienen algunas aristas en común. Esto hace que algunas aristas se

generen al azar, haciendo que el algoritmo en ocasiones se desempeñe similar a una búsqueda aleatoria. Para evitar este problema, Palmer y Kershenbaum (1994) proponen una codificación versátil de árboles de esparcimiento a la cual llaman codificación con prioridades en aristas y vértices (*Link and Node biased*). En esta codificación, un cromosoma es una cadena con pesos asociados a los vértices del grafo y opcionalmente con sus aristas. Esta representación añade temporalmente el peso de cada vértice al costo de todas las aristas a las que el vértice incide, además de añadir los costos que tienen asignadas las aristas; después se utiliza el algoritmo de Prim para encontrar el árbol de esparcimiento máximo de los costos modificados de las aristas, el cual requiere un tiempo de $O(m + n \log n)$ para su codificación y decodificación.

Para esta representación existen las variantes de tomar solamente los pesos de las aristas (prioridad en las aristas o *Link-Biased*), los pesos de las vértices (prioridad en las vértices o *Node-only Biased*), o ambos. En la Figura 8 se muestra un ejemplo de la variante con prioridad en las aristas; esta variante utiliza la fórmula $d'_{ij} = d_{ij} + p\, b_{ij}\, d_{max}$ para calcular el costo modificado de cada arista, en donde d_{ij} es el costo original de la arista que va del vértice i al vértice j, d_{max} el costo máximo de las aristas, $b_{ij} \in (0, 1)$ el elemento correspondiente en el vector de tendencia (*bias vector*), y finalmente, p un parámetro que controla la influencia del vector de tendencia. El parámetro p tiene un fuerte impacto en la estructura del árbol; por ejemplo, si $p = 0$, la importancia del vector de tendecia es nula y el algoritmo funciona como si se corriera el algoritmo de Prim en el grafo original. Cabe mencionar que para cada variante se utiliza una fórmula distinta tomando como base la recientemente descrita. En la Figura 8a se muestra el grafo de entrada; en la Figura 8b se ilustra el grafo con los pesos modificados utilizando la fórmula de prioridad en las aristas; por último, en la Figura 8c se observa el árbol resultante al aplicar la codificación con prioridad en las

aristas. A continuación se muestran los cálculos que se realizan en la Figura 8 con el parámetro $p = 1$:

Vector de pesos originales:

$$d_{AB} = 0,\ d_{AC} = 2,\ d_{AD} = 1,\ d_{BC} = 4,\ d_{BD} = 3,\ d_{CD} = 5$$

Vector de tendencia:

$$b_{AB} = 0.99,\ b_{AC} = 0.87,\ b_{AD} = 0.12,\ b_{BC} = 0.27,\ b_{BD} = 0.03,\ b_{CD} = 0.77$$

Cálculo de pesos modificados:

$$d'_{AB} = 0 + 1 * 0.99 * 5 = 4.95,\ d'_{AC} = 2 + 1 * 0.87 * 5 = 6.35,$$

$$d'_{AD} = 1 + 1 * 0.12 * 5 = 1.6,\ d'_{BC} = 4 + 1 * 0.27 * 5 = 5.35,$$

$$d'_{BD} = 3 + 1 * 0.03 * 5 = 3.15,\ d'_{CD} = 5 + 1 * 0.77 * 5 = 8.85$$

Rothlauf y Gaube (2001) revelaron por medio de una investigación empírica que usar la versión simple de la codificación (prioridad en aristas o en prioridad en vértices) tiende a generar redes similares a estrellas enraizadas y unos pocos individuos dominan la población inicial. Para evitar esto se sugiere utilizar la versión más costosa de la codificación con prioridades en aristas y vértices, aunque solo para el problema *Optimal Communication Spanning Tree*. Una ventaja de esta representación es que cada cadena generada es válida, así que pueden aplicarse operadores de mutación y cruce por posiciones. Una desventaja es que el tamaño del cromosoma es igual al número de aristas en el grafo, es decir que para un grafo completo el tamaño del cromosoma es $n(n-1)/2$. Raidl y Julstrom (2003) proponen una variante a esta codificación en un algoritmo evolutivo, llamada *Weight-Coding*, para el problema de *Degree-Constrained Minimum Spanning Tree*. En *Weight-Coding* los pesos de cada cromosoma se seleccionan inicialmente a partir de una distribución log-normal, y su esquema de tendencia es multiplicativo en lugar de aditivo.

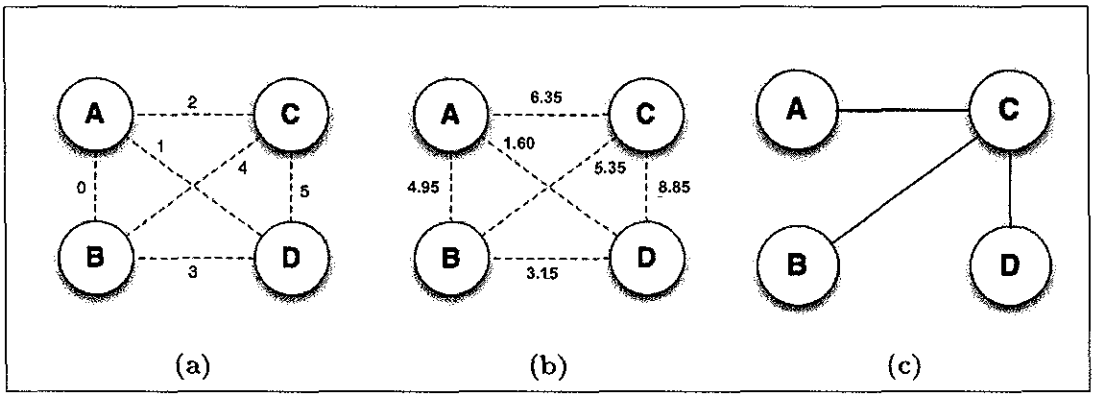


Figura 8: Ejemplo de mapeo para la codificación con prioridades en aristas. a) Grafo inicial; b) Grafo con los pesos modificados; c) Árbol resultante.

Llaves de red aleatorias

Rothlauf *et al.* (2002) propusieron la representación de llaves de red aleatorias (*network random keys o NetKeys*), esta representación es muy similar a la codificación con prioridades en aristas, pero en este caso no se utiliza una fórmula para recalcular los pesos. En esta codificación el cromosoma asigna a cada arista en la red un grado de importancia, que se puede ver como su peso; este peso es un número real en el intervalo $[0, 1]$. Un árbol de esparcimiento se decodifica del cromosoma al añadir aristas de la red a un grafo inicialmente vacío por orden de importancia, ignorando las aristas que introduzcan ciclos, como en el algoritmo de Kruskal. El problema más serio con NetKeys es su costo computacional alto, ya que el cromosoma tiene un tamaño $O(m)$, donde $|m|$ es el número de aristas en el grafo; el tiempo requerido para los operadores de cruce y mutación es $O(m)$ y la codificación es costosa ya que toma $O(m\ log\ m)$. La ventaja es que cualquier secuencia generada con NetKeys es válida, por lo que no es necesario ningún mecanismo de reparación. En la Figura 9 se observa un ejemplo de NetKeys.

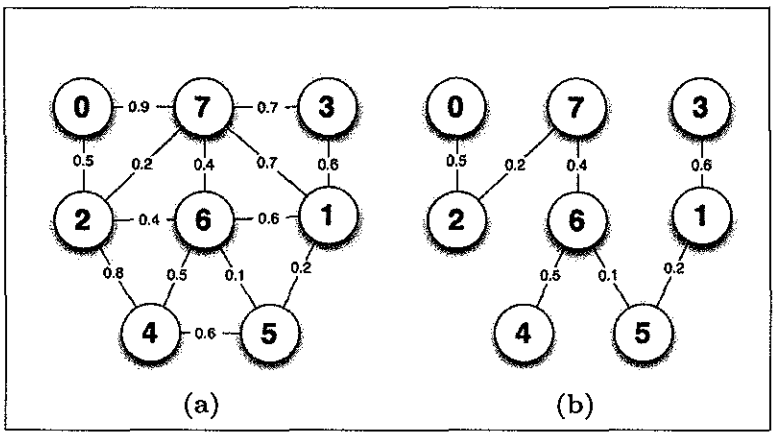


Figura 9: Ejemplo de mapeo para la representación NetKeys.
a) Grafo inicial con pesos asignados a sus aristas; b) Árbol resultante.

Conjunto de aristas

Raidl y Julstrom (2003) propusieron la representación *EdgeSet* o conjunto de aristas; ellos también la compararon con otras representaciones de árboles. Sus resultados muestran que *EdgeSet* es la mejor; esta comparación se puede ver en forma ampliada en la Tabla I de la Página 35. Esta representación es simplemente un vector donde se almacenan todas las aristas que contiene el árbol. Puede implementarse en un arreglo o en una tabla Hash, en los cuales sus entradas son un par de vértices que definen cada arista. Utilizando tablas Hash, la inserción y borrado de aristas individuales requiere tiempo constante, mientras que con arreglos es $O(n)$. Esta codificación requiere un espacio lineal en el número de vértices. Un algoritmo evolutivo requiere una población inicial de diversos cromosomas, así que por lo general se recurre a generar la población de forma aleatoria. Una ventaja para la representación *EdgeSet*, es que utilizan operadores de cruzamiento y mutación utilizando métodos aleatorios de bajo costo y siempre se obtienen soluciones factibles.

Julstrom (2005) utiliza un cruzamiento basado en el algoritmo de Kruskal. Su algoritmo copia las aristas que tienen en común ambos padres en el cromosoma descendiente;

después, sigue seleccionando aleatoriamente las aristas restante de los padres, descartando aquellas que generen ciclos, hasta que el descendiente representa un árbol en todos los vértices; para el caso de la mutación se toma el cromosoma de un padre y con una pequeña probabilidad se sustituye cada arista. En la Figura 10 se muestra un grafo de ejemplo y el *EdgeSet* que lo representa.

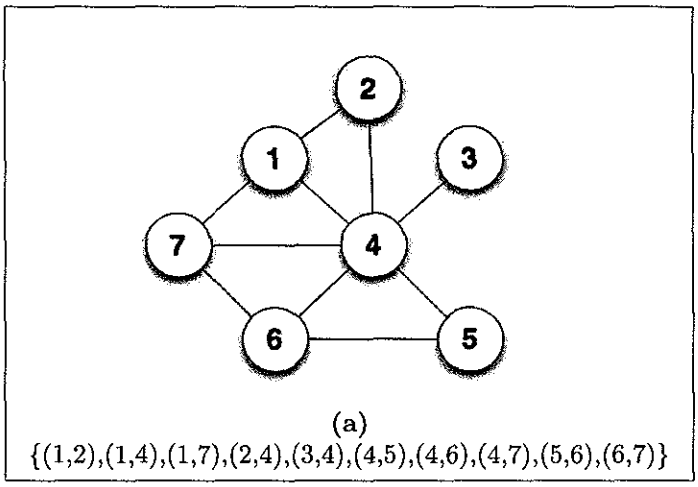


Figura 10: Grafo de ejemplo y el EdgeSet que lo representa.

Dandelion code

Picciotto (1999) propuso la representación dandelion code, la cual debe su nombre a que los árboles generados por Piccioto con esta codificación son parecidos a un diente de león (ver Figura 11).

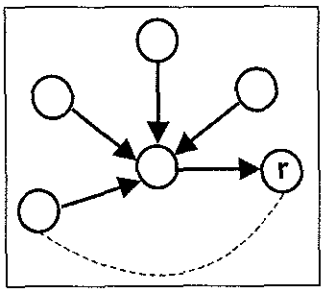


Figura 11: Árbol generado utilizando la codificación para dandelion code de Piccioto.

Una ventaja de esta representación es que mapea el genotipo a un árbol de esparcimiento en $O(n)$ pasos, además tiene características deseables para una representación (unicidad, localidad, factibilidad y herencia persistente), y tiene mejor desempeño que otras representaciones como Prüfer, código blob y NetKeys (Thompson *et al.*, 2007). El dandelion code es un arreglo de sucesores modificado, donde el sucesor del vértice v es aquel vértice que continúa entre la trayectoria de v a la raíz. El algoritmo de codificación utiliza tres funciones; la función $path(v)$, que regresa la trayectoria del vértice v a la raíz del árbol; la función $suc(v)$, que regresa el sucesor de v; y la función booleana $max(v)$, que regresa *true* si v es el vértice con etiqueta mayor de entre él y los vértices en $path(v)$. Para la decodificación, es decir, pasar de dandelion code a un árbol de esparcimiento, se utiliza una relación $\succeq$, donde $c_1 \succeq c_2$ implica que el vértice con etiqueta mayor en el conjunto c_1 es mayor que el vértice con etiqueta más grande en c_2. Cuando se aplican los operadores de variación es necesario aplicar lo que Piccioto denomina *treeSurgery* o cirugía de árbol, esto es generar un grafo a partir del arreglo de sucesores.

El proceso de codificar un árbol a una representación dandelion se explica a continuación: Se supone que los vértices están etiquetados de 0 a $n-1$. Se identifica el arreglo de sucesores, es decir, se obtiene $suc(v)$ por cada vértice en el árbol. Se obtiene el $path(v)$ para $v = n-1$. Se modifica el $path(v)$ generando un ciclo después de cada vértice mayor de derecha a izquierda utilizando la función $max(v)$; por último, se altera el arreglo de sucesores creando los ciclos que se generan en el paso anterior y opcionalmente se aplica *treeSurgery*.

No es tan claro ver una función de biyección para el proceso de codificación de dandelion; sin embargo, la biyección existe y a continuación se describe. Se identifican los ciclos en el arreglo de sucesores. Cada ciclo identificado se reordena en forma de

corrimiento, de tal forma que el máximo elemento aparezca al último. Una vez hecho lo anterior, los ciclos ordenados se concatenan, generando un vector que se llama π. Al principio del vector π se agrega el vértice con etiqueta $n-1$ y al final se agrega la raíz; con esto se obtiene la misma trayectoria $path(v)$ que se tenía en la codificación. Finalmente, se modifica el arreglo de sucesores tomando en cuenta π y opcionalmente se aplica $treeSurgery$. En la Figura 12 se observa un ejemplo de la codificación de un árbol con su arreglo de sucesores al grafo resultante con su dandelion code.

Paulden y Smith (2006b) derivan ocho distintas representaciones basadas en dandelion Code, a las cuales se les denomina *Dandelion-Like*. Su principal diferencia con la codificación original es que se obtiene el menor o el mayor vértice recorriendo el vector π de derecha a izquierda o de izquierda a derecha, entre ellas sobresalen *happy code*; esta última ya la había propuesto Piccioto y debe su nombre a que, comparada con el código blob, es una codificación más 'feliz' ya que es más directa y menos costosa. Otra variante es *rainbow code* propuesta por Paulden y Smith (2004), la cual se utiliza para grafos por capas (*Layered graphs*); un ejemplo de este tipo de grafos se muestra en la Figura 13.

Caminiti y Petreschi (2009) implementaron un algoritmo paralelo para las representaciones basadas en dandelion code; dicha codificación se realizó suponiendo una arquitectura EREW PRAM. El algoritmo toma un tiempo de $O(\log n)$ con $O(n/\log n)$ operaciones, el costo (producto del tiempo y el número de operaciones) es lineal, por lo cual el algoritmo es óptimo. Para tener una implementación eficiente para la decodificación fue necesaria una CREW con tiempo de $O(\log n)$ con $O(n)$ procesadores. Un año después, los mismos autores implementan un algoritmo paralelo para el códgio blob.

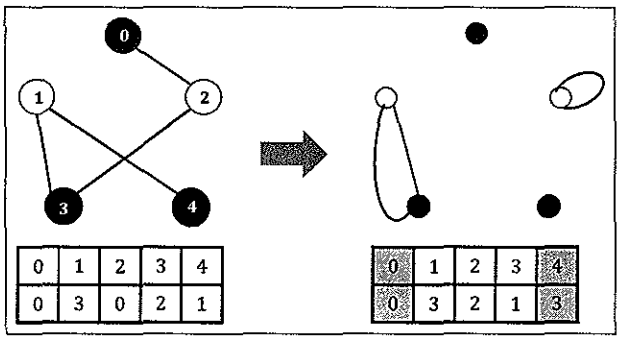


Figura 12: Árbol inicial con su arreglo de sucesores y grafo generado utilizando *treeSurgery* con su arreglo de sucesores modificado.

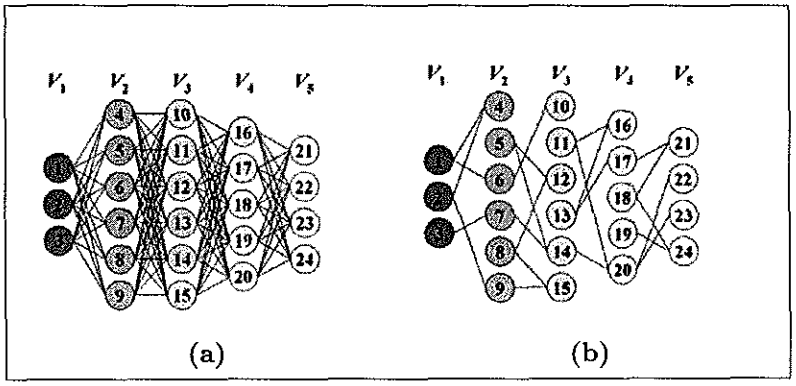


Figura 13: Ejemplo de un grafo por capas.
a) Grafo por capas; b) Árbol de esparcimiento.

Análisis y comparaciones

En la Tabla I se muestra una comparación directa entre las codificaciones mencionadas y las características deseables para una representación, la cual la elaboraron parcialmente Raidl y Julstrom (2003). A la tabla se le agregaron tres codificaciones más dandelion code, happy code y rainbow Code.

II.4.3. Hibridación de algoritmos

Como se menciona anteriormente, una heurística es una técnica para resolver problemas con base en la experiencia, utilizando información para controlar el manejo de la resolución del problema. Una metaheurística es un proceso maestro iterativo que dirige y modifica las operaciones de las heurísticas subordinadas para producir de forma eficiente soluciones de calidad. Una metaheurística puede manipular una solución completa (o incompleta) o una colección de soluciones en cada iteración. La heurística subordinada puede ser un procedimiento de alto o bajo nivel, una búsqueda local simple, o solo un método de construcción. Las metaheurísticas han demostrado ser de gran utilidad para resolver problemas de optimización difíciles en la práctica. Un panorama general de esta área de investigación puede encontrarse en Blum y Roli (2003) y Glover y Kochenberger (2003). Glover (1986) introdujo el término de metaheurística. Este término se refiere a una clase amplia de conceptos de algoritmos para optimización y resolución de problemas.

Las metaheurísticas pueden incluir un procedimiento que use una estrategia para superar la trampa de un óptimo local en espacios de solución complejos. Estos procedimientos pueden utilizar uno o más estructuras para el vecindario, esto es definir movimientos admisibles de una solución a otra. Recientemente se reporta un gran nú-

Tabla I: Comparación entre distintas propiedades que deben tener las representaciones de árboles de esparcimiento.

* Representaciones agregadas por el autor a la tabla de Raidl y Julstrom (2003).

Representación	Espacio	Tiempo	Fact.	Uni.	Tend.	Loc./Her.	Restr.	Graf. Disp
Vector Característico	$O(m)$	$O(m)$	Peor	Si	No	Alta	Promedio	Buena
Vector Predecesor	$O(n)$	$O(n)$	Pobre	Si	No	Alta	Promedio	Pobre
Número de Prüfer	$O(n)$	$O(n \log n)$	Si	Si	No	Baja	Pobre	Pobre
Blob Code	$O(n)$	$O(n^2)$	Si	Si	No	Promedio	Pobre	Pobre
Link and Node Biased	$O(m+n)$	$O(m+n \log n)$	Si	Si	Alta	Promedio	Buena	Buena
Node-only Biased	$O(n)$	$O(m+n \log n)$	Si	No	Alta	Promedio	Bueno	Buena
Llaves de Red Aleat.	$O(m)$	$O(m \log m)$	Si	Si	Baja	Alta	Bueno	Buena
EdgeSet	$O(n)$	$O(n)$	Si	Si	Depende	Mejor	Bueno	Buena
Dandelion Code*	$O(n)$	$O(n)$	Si	Si	Si	Alta	Bueno	Buena
Happy Code*	$O(n)$	$O(n)$	Si	Si	Si	Alta	Bueno	Buena
Rainbow Code*	$O(n)$	$O(n)$	Si	Si	Si	Alta	Promedio	Buena

Fact.: Factibilidad

Uni.: Unicidad

Tend.: Tendencia

Loc./Her.: Localidad y herencia persistente

Restr.: Restricciones

Graf. Disp.: Grafos dispersos

mero de algoritmos que no siguen estrictamente los conceptos de una metaheurística tradicional, si no que combinan varias ideas de algoritmos, incluso algunas veces de otro campo de la metaheurística tradicional. Estos enfoques se conocen comúnmente como metaheurísticas híbridas. Así como para una metaheurística en general, existen distintas percepciones de lo que es realmente una metahuerística híbrida. El término de híbrido tiene distintas definiciones:

- Algo heterogéneo en origen y descomposición.
- Algo que tiene dos diferentes tipos de componentes que desempeñan esencialmente la misma función.
- Descendientes resultado del cruzamiento de diferentes entidades, i.e. diferentes especies.
- Algo de origen mixto y composición.

La motivación detrás de la hibridación de diferentes conceptos algorítmicos es obtener sistemas con un mejor desempeño que exploten y unan sus ventajas de estrategias puras individuales, es decir, tales híbridos se cree se benefician de la sinergia. El número incremental de aplicaciones reportadas de metaheurísticas híbridas y dedicadas a los eventos científicos, documenta la popularidad, éxito e importancia de esta línea de investigación. En realidad, ahora parece que el seleccionar un enfoque híbrido es determinante para lograr un alto desempeño al resolver problemas de gran dificultad. Incluso, la idea de hacer híbrida una metaheurística no es nueva, sino que se remonta al origen de las propias metaheurísticas. Al principio, sin embargo, los híbridos no eran tan populares ya que distintas áreas fuertemente separadas y comunidades competitivas de investigadores consideraban su 'clase' favorita de metaheurística 'generalmente mejor'

y seguían una filosofía específica de forma muy dogmática. Por ejemplo, la comunidad de cómputo evolutivo creció de forma relativamente aislada y seguía estrictamente el enfoque biológico. Es debido a el teorema del no almuerzo gratis que esta situación cambió y las personas reconocieron que no puede existir una estrategia de optimización general que sea mejor globalmente que cualquier otra. De hecho, para resolver un problema de forma más efectiva, se requiere casi siempre de un algoritmo especializado que esté compuesto de partes adecuadas.

Capítulo III

Definición del problema

III.1. Problema de asignación de guardaespaldas

Fajardo y Fernández (2010) propusieron el problema de asignación de guardaespaldas. La definición formal del problema se especifica a continuación:

Entrada: Un grafo conectado no dirigido $G = (V, E)$ con sus vértices etiquetados, donde $|V| = n$, un vértice especial r y una partición de $V = V_1 \cup V_2 \cup V_3$, tal que $V_1 = \{r\}$, V_2 es el conjunto de vértices blancos y V_3 es el conjunto de vértices negros.

Salida: Un árbol de esparcimiento $T = (V, E_T)$, tal que el conjunto de vértices $V_2 \cup V_3$ maximice la siguiente función:

$$Ganancia(T) = \sum_{v \in V_2 \cup V_3} ganancia(v)$$

donde:

$ganancia(v) = dist_r(v)$ si $v \in V_3$

$ganancia(v) = n - (dist_r(v))$ si $v \in V_2$

La distancia $dist_r(v)$ se define como el número de aristas entre el vértice v y la raíz en el árbol de esparcimiento T.

El tamaño del espacio de búsqueda para este problema, depende de la densidad del grafo. Para un grafo completo el tamaño del espacio de búsqueda es el número de árboles posibles, si se tiene n vértices etiquetados pueden conectarse de n^{n-2} maneras, es decir, se pueden formar n^{n-2} árboles de un grafo completo. Nótese que este es el peor caso, ya que si se tiene un árbol disperso el espacio de búsqueda se reduce considerablemente.

III.2. Antecedentes

En el trabajo de Fajardo y Fernández (2010) se propone un algoritmo determinístico que provee soluciones aproximadas al PAG, con este algoritmo se obtienen buenos resultados, pero no siempre se encuentra el óptimo global ya que en ocasiones la solución generada por el algoritmo es un óptimo local. Uno de los objetivos principales planteados en esta investigación es generar un algoritmo evolutivo que mejore los resultados del algoritmo determinístico.

III.2.1. Algoritmo determinístico

El algoritmo de Fajardo y Fernández (2010) funciona por medio de un calendarizador central que tiene como función seleccionar el vértice que va a realizar una acción de optimización; ellos implementaron distintos tipos de calendarizadores. Los vértices realizan alguna acción dependiendo de su preferencia; su color determina su preferencia o algún enfoque de operación distinto. Fajardo y Fernández (2010) utilizan dos enfoques en sus algoritmos, un enfoque cooperativo, donde todos los vértices tratan de mejorar la ganancia global de la solución, mientras que con el enfoque no cooperativo cada vértice le da preferencia a mejorar su ganancia individual por encima de la ganancia global; ellos determinan que el enfoque cooperativo es con el que se obtienen mejores resultados en forma general. Como se mencionó anteriormente, sus resultados muestran que su algoritmo llamado CBAP (Centralized Bodyguard Allocation Problem) obtiene buenos resultados, pero frecuentemente se queda estancado en un óptimo local.

III.3. Óptimo global

Con la finalidad de poder determinar qué tan bueno es el desempeño de los algoritmos evolutivos para el PAG, se buscó una forma de calcular la máxima ganancia que se puede obtener dado un grafo completo de entrada (i.e. un grafo con todas las aristas posibles), para un número dado de vértices. Es decir, se contestó la siguiente pregunta: ¿Cuál es la máxima ganancia global que se puede obtener para el PAG suponiendo que su entrada es un grafo completo?. Los algoritmos evolutivos se comportan de manera cooperativa, ya que solo se optimiza la ganancia global. En Fajardo y Fernández (2010) se determina que la solución óptima para un algoritmo con enfoque cooperativo es una cadena donde al principio se encuentra la raíz, seguida de todos los vértices blancos y por último todos los vértices de color negro; para el enfoque no cooperativo la solución es parecida a una estrella enraizada, donde los vértices blancos están conectados directamente a la raíz y los vértices negros forman una cadena conectada a un vértice blanco. Se analizó un caso base de una solución para un enfoque cooperativo, con un árbol de 5 vértices, como se muestra en la Figura 14. Se sabe que la ganancia de un vértice blanco v es el número de vértices en el árbol menos la distancia del vértice v a la raíz $(n - dist_r(v))$, mientras que el de un vértice negro v es solamente la distancia a la raíz $(dist_r(v))$; por lo que al tener una solución como en la Figura 14 (enfoque cooperativo) y sumar la ganancia de todos los vértices blancos y todos los vértices negros se observa que se obtienen los mismos valores de ambas sumatorias. Es importante notar que para el vértice blanco más cercano a la raíz, se obtiene exactamente la misma ganancia que con el vértice negro más alejado. Para este ejemplo específico, la ganancia para el más cercano es $(n - dist_r(v) = 5 - 1)$ y que es igual al negro más lejano $(dist_r(v) = 4)$; esta igualdad se cumple a su vez para el segundo vértice blanco

más cercano a la raíz y el segundo vértice negro más lejano, y asi sucesivamente. Generalizando para cualquier caso, en donde el número total de vértices n sea impar y la cantidad de vértices blancos y negros sea la misma $\frac{n-1}{2}$, entonces, por cada tipo de vértice se obtendrá una sumatoria $\sum_{i=1}^{\frac{n-1}{2}} (n-i)$; para ambos tipos de vértices, la ganancia total del árbol es $2\sum_{i=1}^{\frac{n-1}{2}} (n-i)$. A continuación se obtiene una fórmula cerrada para esta sumatoria. Primero se descomponen los términos de la sumatoria como se muestra en la Ecuación 1.

$$GananciaMax = 2\sum_{i=1}^{\frac{n-1}{2}} (n-i) = 2\left[\sum_{i=1}^{\frac{n-1}{2}} n - \sum_{i=1}^{\frac{n-1}{2}} i\right]. \tag{1}$$

Dado que la primer sumatoria no depende de i, ésta es igual al número de veces a sumarse por n, esto es $n(\frac{n-1}{2})$; mientras que la segunda sumatoria es la serie aritmética por lo cual es igual a $k(\frac{k+1}{2})$, donde $k = \frac{n-1}{2}$; sustituyendo esto en la sumatoria y desarrollando se obtiene la Ecuación 2.

$$GananciaMax = 2\left[\sum_{i=1}^{\frac{n-1}{2}} n - \sum_{i=1}^{\frac{n-1}{2}} i\right] = 2\left[n\left(\frac{n-1}{2}\right) - \frac{(\frac{n-1}{2})(\frac{n-1}{2}+1)}{2}\right]. \tag{2}$$

Eliminando el 2 que divide y multiplica, se obtiene la Ecuación 3.

$$GananciaMax = n(n-1) - \left(\frac{n-1}{2}\right)\left(\frac{n-1}{2}+1\right) = n(n-1) - \left(\frac{n-1}{2}\right)\left(\frac{n+1}{2}\right). \tag{3}$$

Desarrollando y agrupando se obtiene la Ecuación 4.

$$GananciaMax = n\,(n-1) - \left(\frac{(n-1)\,(n+1)}{4}\right) = (n-1)\left(n - \left(\frac{n+1}{4}\right)\right). \quad (4)$$

Por último reduciendo la expresión se obtiene la Ecuación 5.

$$GananciaMax = (n-1)\left(\frac{4n-n-1}{4}\right) = \frac{(n-1)\,(3n-1)}{4}. \quad (5)$$

La ganancia máxima de un árbol en forma de trayectoria dado el número de vértices n, se puede expresar con la Ecuación 6. Note que al evaluar la expresión de la Ecuación 6 con $n = 5$ se obtiene que la ganancia máxima es 14.

$$GananciaMax = (n-1)\left(\frac{4n-n-1}{4}\right) = \frac{(n-1)\,(3n-1)}{4}. \quad (6)$$

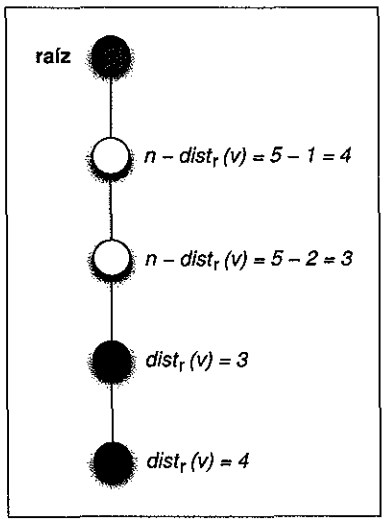


Figura 14: Solución óptima para el PAG, dado un grafo de 5 vértices con un enfoque cooperativo.

III.4. Problemas similares

En esta sección se analizan problemas similares al PAG en los cuales sus entradas son grafos y sus soluciones son árboles de esparcimiento; además, se discuten cuáles trabajos se han intentado solucionar por medio de cómputo evolutivo y con cuál representación se obtuvieron mejores resultados. El objetivo de analizar los problemas similares es el de encontrar datos relevantes para la investigación. A continuación se describen dichos problemas:

- **Árbol de Steiner** ***(Steiner tree)*****:** Dado un conjunto de vértices terminales y un conjunto de vértices Steiner en un plano, se intenta encontrar el árbol de esparcimiento mínimo que conecte todos los nodos terminales permitiendo agregar algunos Steiner. Existen tres variantes a este problema. El primero es el problema del árbol de Steiner Euclidiano *(Euclidean Steiner tree problem, ESTP)*; en él se utiliza el concepto de distancia Euclideana entre dos puntos (norma). La segunda variante es el problema del árbol de Steiner rectilíneo *(rectilinear Steiner tree problem, RSTP)*; en él se utiliza el concepto de distancia rectilínea, es decir la sumatoria del valor absoluto de la diferencia de las coordenadas entre dos puntos. Por último, el problema de encontrar el Árbol de Steiner en un grafo (GSTP). Yang (2006) aborda el ESTP, donde define un individuo como el conjunto de posiciones de los puntos Steiner en el plano. Cada individuo puede tener longitud diferente dependiendo del número de nodos Steiner en el individuo y representa un árbol de Steiner. La aptitud del individuo corresponde a la longitud del MST que se puede construir al usar los vértices terminales y los vértices Steiner en el individuo utilizando el algoritmo de Prim. Kumar *et al.* (2005) abordaron el ESTP tomando las entradas o los puntos de referencia (*benchmarks*) utilizados

anteriormente por Raidl.

- **Árbol de esparcimiento minimo con restricción de grado *(Degree Constrained Minimum Spanning Tree, DCMST)*:** En un grafo, el grado de un vértice es el número de aristas adyacentes a él; el grado del grafo es el máximo grado de sus vértices. Dado un grafo G, no dirigido, en este problema se consideran todos los árboles de esparcimiento cuyos grados no excedan la cota $d \geq 2$; de entre éstos, el árbol con el costo mínimo es el DCMST. Este Problema es NP-difícil y lo discuten en distintos trabajos tanto en su forma tradicional de un solo objetivo (Raidl y Julstrom, 2003), (Gottlieb *et al.*, 2001), (Knowles y Corne, 2000) como Multiobjetivo (Bui y Zrncic, 2006), (Knowles y Corne, 2001b), (Knowles y Corne, 2001a).

- **Árbol de esparcimiento para comunicación optima (*Optimum comunication spanning tree, OCST*):** Para este problema se busca un árbol de esparcimiento que satisfaga ciertos requerimientos de comunicación entre distintos vértices y que además tenga el menor costo. Sea T un árbol de esparcimiento de un grafo $G = (V, E)$ conectado, no dirigido y con pesos asignados a sus aristas. Cualquier par de vértices $u, v \in V$ están conectados por una única trayectoria en T; la suma de los pesos de las aristas en la trayectoria es el costo de la trayectoria de ruteo $c_T(u, v)$. El costo de ruteo total $C(T)$ de T es la suma de los costos de ruteo de todas las trayectorias entre cada par de vértices distintos en T. El costo de una arista es resultado del producto de la tasa de flujo y el costo de ruteo por unidad de flujo. El objetivo es minimizar el costo de la Ecuación 8.

$$C(T) = \sum_{u,v \in V} c_T(u, v), \tag{7}$$

$$Costo = C(T) \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} r_{u,v}, \tag{8}$$

donde $r_{u,v}$ es el requerimiento entre los vértices u y v. Por lo general se utiliza una matriz de costos, donde se asigna un costo entre cualquier par de vértices, haciendo que la matriz de costos sea completa. Este problema es un problema NP-difícil. Palmer y Kershenbaum (1994) proponen *Link and Node Biased* y lo utilizan para intentar resolver este problema, mientras que Gottlieb *et al.* (2001) demuestran que Prüfer es una codificación deficiente para este tipo de problemas. Raidl y Julstrom (2003) utilizan entradas de este problema para establecer comparaciones entre distintas representaciones de árboles; los resultados se muestran en la Tabla I de la Página 35. Julstrom (2005) implementa código blob utilizando el número de Prüfer y lo compara con la codificación EdgeSet, concluyendo que obtiene resultados similares. Thompson *et al.* (2007) comparan varias representaciones (Prüfer, código blob, network random keys y dandelion code) para generar soluciones a este problema, siendo dandelion la que obtiene mejores resultados.

- **Árbol máximo único *(One max tree, OMT)*:** Se utiliza para probar el desempeño de algoritmos de optimización de árboles para diseño topológico. Para este problema se establece una solución óptima de antemano. Para el cálculo de la aptitud de los individuos se utiliza una noción de la distancia de Hamming, que se define como la distancia entre dos árboles T_1 y T_2, esto es, el número de aristas que pertenecen a T_1 pero no a T_2 (o de forma equivalente, el número de aristas que se requieren intercambiar para tranformar T_1 a T_2). Se quiere minimizar la distancia entre la solución óptima y el individuo, esta función se muestra en la Ecuación 9. Sea a un individuo y opt el óptimo, $d_{a,opt}$ denota la aptitud del individuo a, esto

es, la distancia entre a y el árbol de esparcimiento óptimo.

$$d_{a,opt} = \frac{1}{2}\sum_{i=1}^{n-1}\sum_{j=0}^{i-1}\left|l_{ij}^{a} - l_{ij}^{opt}\right|, \tag{9}$$

donde l_{ij}^{a} es 1 si la arista del vértice i al vértice j existe en el árbol a y 0 de lo contrario; $d_{a,opt} \in \{0, 1, ..., n-2\}$, de tal forma que si el individuo a sólo tiene una arista en común con el óptimo, su aptitud es $d_{a,opt} = n - 2$; pero si a y el óptimo son el mismo árbol, entonces $d_{a,opt} = 0$. Para este problema, Rothlauf *et al.* (2002) utilizan la representación *network random keys*, mientras que Rothlauf y Goldberg (2003) implementan *Link-Biased*, y Thompson *et al.* (2007) comparan las codificaciones de Prüfer, código blob, *network random keys* y dandelion, siendo esta última la que mejores resultados obtiene.

- **Árbol de esparcimiento minimo con restricción en el número de hojas *(The leaf-constrained minimum spanning tree, LCMST)*:** Para este problema se busca un árbol de esparcimiento que tenga al menos l hojas, donde l es un entero, $2 \leq l \leq n-1$, y además sea el árbol de esparcimiento de menor peso. Este problema es NP-difícil. Julstrom (2004) compara una estrategia egoísta con algoritmos genéticos que utilizan la codificación de Prüfer y Blob Code.

- **Árbol de esparcimiento con costo de ruteo mínimo *(Minimum routing cost spanning tree, MRCST)*:** Sea T un árbol de esparcimiento en un grafo conectado, no dirigido y con pesos asignados a las aristas. Para cualquier par de vértices $u, v \in V$ conectados por una sola trayectoria en T, la suma de los pesos de las aristas es el costo de la trayectoria $c_T(u, v)$. El costo del ruteo o longitud total de la trayectoria $C(T)$ de T es la suma de los costos de ruteo de las trayectorias en

T entre cualquier par de vértices. El MRCST busca el árbol en G con el mínimo costo de ruteo $C(T)$. Este problema es un caso específico de OCST. En el trabajo de Julstrom (2005) se concluye que Blob Code genera buenas soluciones para este problema.

- **Árbol de esparcimiento de etiquetado mínimo *(Minimum labeling spanning tree, MLST)*:** Dado un grafo G conectado, no dirigido, cuyas aristas están etiquetadas (coloreadas), el MLST busca un árbol de esparcimiento en G con el menor número de etiquetas (colores) distintos. Este problema está motivado por aplicaciones en diseño de redes de comunicación. Este problema es NP-difícil, Nummela y Julstrom (2006) generan soluciones aproximadas al utilizar algoritmos genéticos.

- **Problema de transporte de carga fija *(Fixed charge transportation problem, FCTP)*:** En la práctica es un problema de transporte en el sector de distribución de productos. Dadas m plantas (orígenes) y n consumidores (destinos), el problema se puede formular como se expresa en la Ecuación 10:

$$costo = Min\left\{\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij}\sum_{i=1}^{m} f_i\right\} \tag{10}$$

sujeto a:

$$\sum_{j=1}^{n} x_{ij} \leq a_i, i = 1, 2, ..., m \qquad \sum_{i=1}^{m} x_{ij} = b_j, j = 1, 2, ..., n \qquad x_{ij} \geq 0, \forall i, j$$

con:

$$f_i(x) = \begin{cases} 1, \ si \ \ x_{ij} > 0 \\ 0, \ de \ otra \ forma \end{cases} \tag{11}$$

donde $i = \{1, 2, ..., m\}$ son los índices de los m orígenes; $j = \{1, 2, ..., n\}$ son los índices de los n destinos; x_{ij} = la cantidad que se transporta del origen i al destino j; c_{ij} = el costo por unidad que se transporta del origen i al destino j; f_i = el costo fijo asociado con el origen i; a_i = la máxima cantidad disponible en el origen i; b_j = la demanda del destino j. En este modelo, la primera restricción implica que la cantidad total que se transporta de la fuente i no debe ser mayor a a_i; la segunda restricción indica que la cantidad total que se transporta de todas las fuentes i debe satisfacer la demanda del destino b_j. La última restricción indica que la cantidad que se transporta x_{ij} siempre debe ser mayor que 0 para cualquier origen o destino. Paulden y Smith (2004), Paulden y Smith (2006a) afirman que la variante de dandelion para grafos bipartitos *Rainbow Code* es útil para este tipo de problemas.

Una vez descritas las distintas representaciones para árboles en la Sección II.4, así como la variedad de problemas en los cuales se utilizan técnicas evolutivas con este tipo de representaciones en esta sección. Se concluye que las representaciones que presentan mejores características y además mayor factibilidad de generar buenos resultados para el PAG son dandelion, network random keys y EdgeSet.

Capítulo IV

Generación de casos de prueba

En este capítulo se describen y analizan distintos métodos para generar casos de prueba o entradas válidas para el PAG. Una entrada del PAG debe ser un grafo conectado no dirigido con sus vértices etiquetados, como se describe en la Sección III.1. Existen muchos modelos para generar grafos aleatorios, en este capítulo se describen algunos que se utilizan para obtener un conjunto de entradas lo suficientemente variadas para evaluar si el algoritmo obtiene buenos resultados ante todas ellas. Antes de describir los modelos, es necesario establecer algunas definiciones. Un grafo $G = (V, E)$, donde $|V| = n$ es el número de vértices y $|E| = m$ es el número de aristas; este grafo es completo si $m = n(n-1)/2$; esto es, que para cualquier par de vértices $u, v \in V$, $(u, v) \in E$. Un grafo aleatorio se puede obtener al conectar un conjunto de n vértices aislados por medio de aristas insertadas entre ellos de forma aleatoria. Un grafo no dirigido es aquel que sus aristas no tienen asignada una dirección en particular, es decir la conexión es de un vértice hacia al otro y viceversa. Dos grafos $G = (V, E)$ y $G' = (V', E')$ son isomorfos si existe una biyección $f : V \to V'$ tal que $(u, v) \in E$ si y solo si $(f(u), f(v)) \in E'$. En otras palabras, los vértices de G se pueden reetiquetar para que sean los vértices de G', manteniendo las aristas correspondientes en G y G'. Un multigrafo es aquel grafo que tiene multiples aristas entre dos vértices. Un autociclo es cuando un vértice tiene una arista que apunta a si mismo.

IV.1. Modelos para generar grafos aleatorios

Se utilizó la librería igraph (Csardi y Nepusz, 2006) para la generación de entradas con el lenguaje de programación R. Los siguientes modelos que ya están implementados en la librería y son los que se utilizan para generar las entradas para la etapa de experimentación. Un grafo conectado es aquel en el que desde cualquier vértice se puede llegar a otro por medio de las aristas del grafo, es decir no existen vértices aislados. Se verificó que los grafos generados fueran conectados, no dirigidos, no isomorfos entre si, no multígrafos y sin autociclos.

IV.1.1. Modelo Barabási-Albert

En una red libre de escala, algunos vértices estan altamente conectados, es decir, poseen un gran número de aristas a otros vértices, aunque el grado de conexión de la mayoría de los vértices es bastante bajo. El modelo Barabási-Albert (BA) utiliza un algoritmo que sirve para generar redes aleatorias complejas libre de escala, empleando un mecanismo denominado conexión preferencial. El modelo propuesto por Barabasi y Albert (1999), se basa en las siguientes observaciones de redes reales como la Internet: las redes se expanden de forma continua al agregarse nuevos vértices y los nuevos vértices se conectan preferentemente a los vértices que tienen conexiones múltiples. El modelo se describe a continuación. La construcción del grafo comienza con un conjunto de n vértices conectados aleatoriamente, donde $n \geq 2$ y el grado de cada vértice debe ser al menos 1. Los nuevos vértices se agregan a la red uno a uno. Cada vértice se conecta a m vértices de la red con una probabilidad que es proporcional al número de enlaces que poseen los vértices de la red, es decir, los nuevos vértices se enlazan preferentemente con los vértices con más aristas. Formalmente, la probabilidad p_i de que un nuevo vértice

se conecte con i está dada por la Ecuación 12, donde k_i es el grado del vértice i.

$$p_i = \frac{k_i}{\sum_j k_j}. \tag{12}$$

En otras implementaciones del modelo Barabási-Albert se agrega un parámetro más que determina la atracción de los vértices sin aristas adyacentes, es decir, determina qué tan atractivo es para un vértice nuevo conectarse con un vértice con un bajo grado de conectividad. En la Figura 15 se observan un par de grafos generados con el modelo Barabási-Albert, ambos tienen 51 vértices. Para el grafo de la Figura 15a se fijaron los parámetros de una conexión preferencial de 1 y para la Figura 15b una conexión preferencial de 2, en ambos se fijó en 4 el número de aristas por cada nuevo vértice agregado.

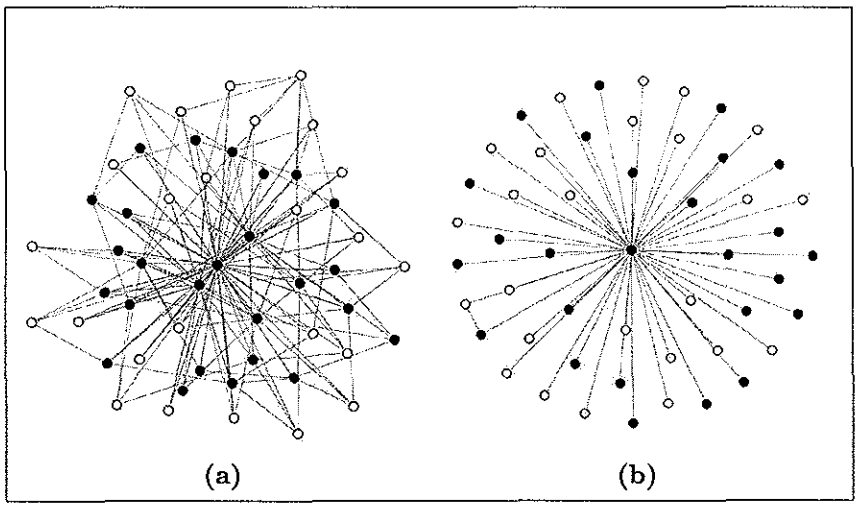


Figura 15: Grafos aleatorios generados con el modelo Barabási-Albert.
a) Grafo con conexión preferencial lineal; b) Grafo con conexión preferencial cuadrática.

IV.1.2. Modelo Erdös Rényi

Erdös y Rényi (1959) proponen un modelo muy utilizado para generar redes aleato-

rias. Existen dos variantes de este modelo, la primera llamada $G(n,p)$ o $G(n,P(e)=p)$, donde el grafo a generar tiene n vértices y cada posible arista del grafo tiene una probabilidad de p de incluirse; en este método, el grafo se va construyendo al conectar vértices de forma aleatoria. Cada arista se incluye en el grafo con probabilidad p independientemente de la inclusión de cualquier otra arista. En esta variante, el parámetro p puede verse como una función de peso; comforme p aumenta de 0 a 1, el modelo tiende a generar grafos con más aristas (grafos densos) y al disminuir p, se generan grafos con pocas aristas (grafos dispersos). La segunda variante de este modelo se llama $G(n,m)$, donde de forma similar, n es el número de vértices del grafo a generar y m es el número de aristas que incluirá el grafo resultante; en este caso se seleccionan m aristas del conjunto de aristas posibles $\left(\frac{n(n-1)}{2}\right)$ para n vértices, es decir, el grafo se va construyendo al ir conectando los vértices de forma aleatoria, hasta tener m conexiones distintas. En la Figura 16 se observan un par de grafos con 51 vértices generados con las variantes del modelo Erdös Rényi $G(n,m)$ y $G(n,p)$, respectivamente.

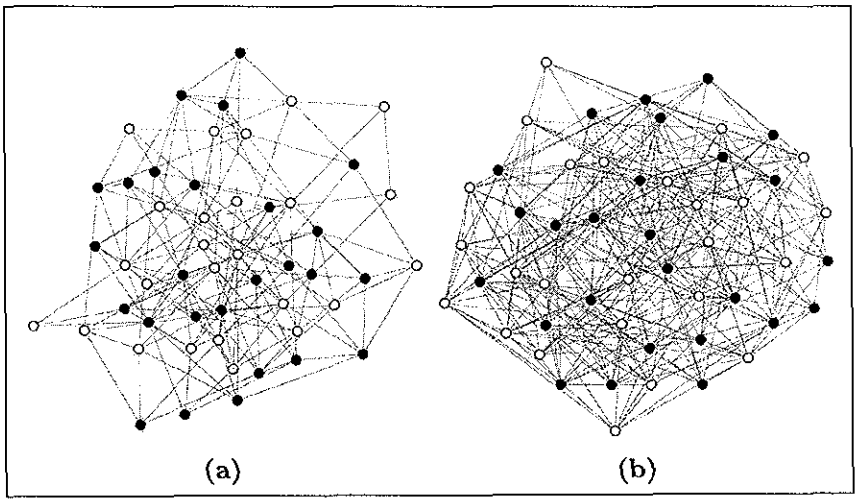


Figura 16: Grafos aleatorios generados con el modelo Erdös Rényi.
a) Grafo con la variante $G(n,m)$ con $m = 162$; b) Grafo con la variante $G(n,p)$ con $p = 0.25$.

IV.1.3. Modelo evolutivo estocástico

En el modelo evolutivo estocástico, Kumar *et al.* (2000) intentan simular una evolución estocástica del grafo; el grafo evoluciona por medio de n pasos de tiempo discreto. El modelo recibe como parámetro el número de vértices n y el número de aristas m a agregar por cada paso. En cada paso un nuevo vértice se agrega al grafo y se crean m nuevas aristas. Si el parámetro *citation* es falso, estas aristas conectan dos vértices seleccionados de forma aleatoria; si es verdadero, las aristas conectan el nuevo vértice a un vértice existente aleatoriamente seleccionado. Nótese que este modelo podría generar grafos con componentes desconectados; para evitar esto, en el proceso de experimentación se utilizó solamente con el parámetro *citation* verdadero. Además, este método puede generar multigrafos (con aristas repetidas), por lo cual se utiliza un proceso para eliminar las aristas duplicadas. En la Figura 17 se muestran un par de grafos de 51 vértices, utilizando $m = 3$ y $m = 4$, respectivamente; en ambos grafos las variables *citation* son verdaderas.

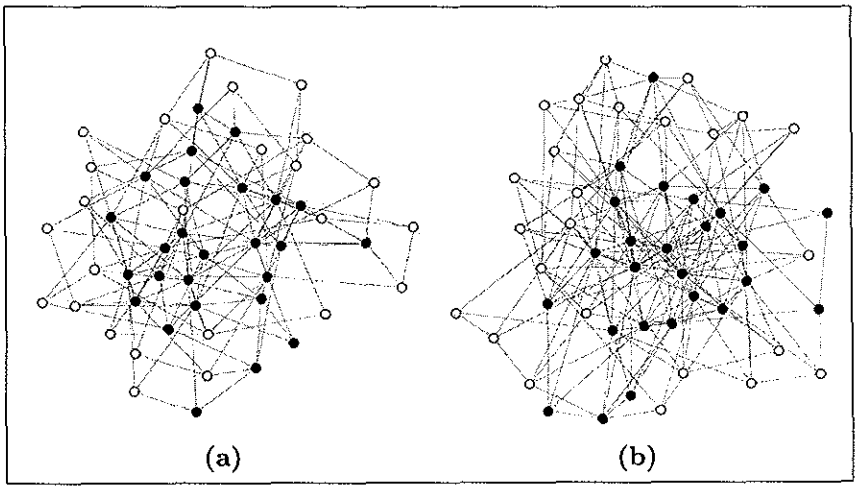


Figura 17: Grafos aleatorios generados con el modelo evolutivo estocástico.
a) Grafo generado con el parámetro $m = 3$; b) Grafo generado con el parámetro $m = 4$.

IV.1.4. Modelo geométrico

Pisanski y Randić (1998) propusieron el modelo geométrico. Este modelo recibe como parámetros un radio *radius*, un valor Booleano t que indica si el plano es un toroide o un plano cuadrado y el número de vértices n. Para la construcción del grafo, primero se colocan n puntos de forma aleatoria en un plano cuadrado o toroidal, dependiendo de la variable t; estos puntos corresponden a los vértices del grafo. Dos vértices se conectan con una arista no dirigida si están cerca uno con otro bajo la norma Euclidiana dada por *radius*. En la Figura 18 se observan dos grafos de 51 vértices con parámetros de $radius = 0.2$, $t = 0$ y $radius = 0.2$, $t = 1$, respectivamente.

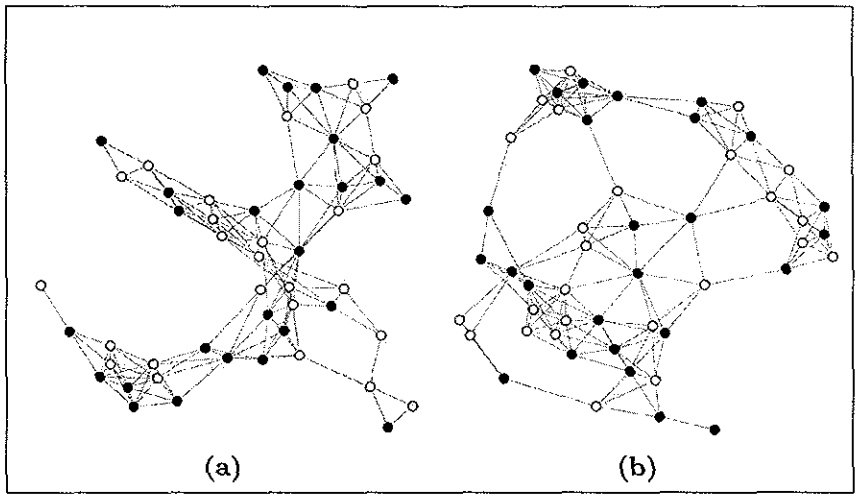


Figura 18: Grafos aleatorios generados con el modelo geométrico.
a) Grafo generado con los parámetros $radius = 0.2$ y $t = 0$; b) Grafo generado con los parámetros $radius = 0.2$ y $t = 1$.

IV.1.5. Modelo grado de adyacencia

El modelo grado de adyacencia se utiliza cuando se quiere crear un grafo con un determinado grado en sus vértices. Para la construcción de grafos, este modelo recibe por parámetros un vector con la cota máxima de los grados d de cada uno de los vértices,

además de especificar el método a utilizar y el número de vértices n a crear. Existen dos tipos de métodos a utilizar, el *simple* y el *vl* (Viger y Latapy, 2005). Para el caso del método *simple*, a cada vértice i se le asigna un número de aristas d_i y después se unen los extremos libres, respetando siempre las cotas en d. Para el método *vl* se contruye un grafo posiblemente conectado, respetando las cotas en d, después se unen los componentes y se desordenan utilizando el método Montecarlo. Cabe destacar que la suma de los grados en d debe ser par debido a que se genera un grafo conectado no dirigido. En la Figura 19 se muestran dos grafos de 51 vértices con una cota $d = 6$ para todos los vértices y utilizando el método *simple* y *vl*, respectivamente.

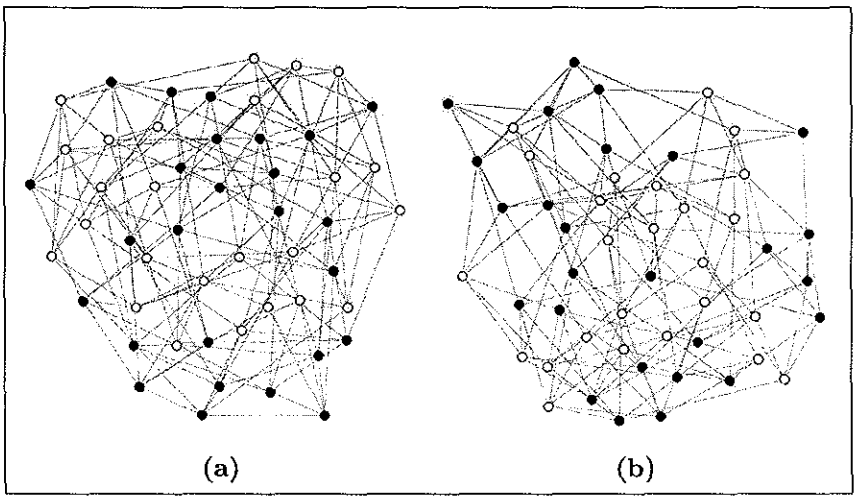


Figura 19: Grafos aleatorios generados con el modelo grado de adyacencia. a) Grafo generado con el método simple y $d = 6$; b) Grafo generado con el método *vl* y $d = 6$.

IV.1.6. Modelo Watts Strogatz

El modelo Watts Strogatz se utiliza ampliamente para la creación de redes de mundo pequeño, esto quiere decir que las distancias medias entre cualquier par de vértices suele ser pequeña. El modelo toma el nombre de una investigación realizada por Watts y

Strogatz (1998). El algoritmo recibe como parámetros el número de dimensiones *dim*, el tamaño *size* de cada dimensión, el número de vecinos *nei* y una probabilidad p. La construcción del grafo inicia al generar una rejilla de tamaño *size* por cada dimensión *dim*, es decir, que se genera una rejilla con un número de vértices $n = size^{dim}$. En el enfoque original solamente se generaba una rejilla en forma de anillo de una sola dimensión, con un número de vértices predefinido. Después de crear la rejilla, las aristas se reconfiguran aleatoriamente con probabilidad p. Note que por este último paso se pueden generar grafos con ciclos y multigrafos, por lo cual se utiliza un mecanismo de reparación del grafo generado. En la Figura 20 se observan dos ejemplos de grafos generados con este modelo; el de la Figura 20a con parámetros $size = 4$, $dim = 2$, $nei = 1$ y $p = 0$, por lo cual su número de vértices es 16; el de la Figura 20b con parámetros $size = 7$, $dim = 2$, $nei = 2$ y $p = 0.7$, por lo que el número de vértices total generado es 49 y al tener una probabilidad de reconfiguración distinta de 0 se puede apreciar que sus vértices no tienen una cantidad uniforme de aristas como en el primer caso.

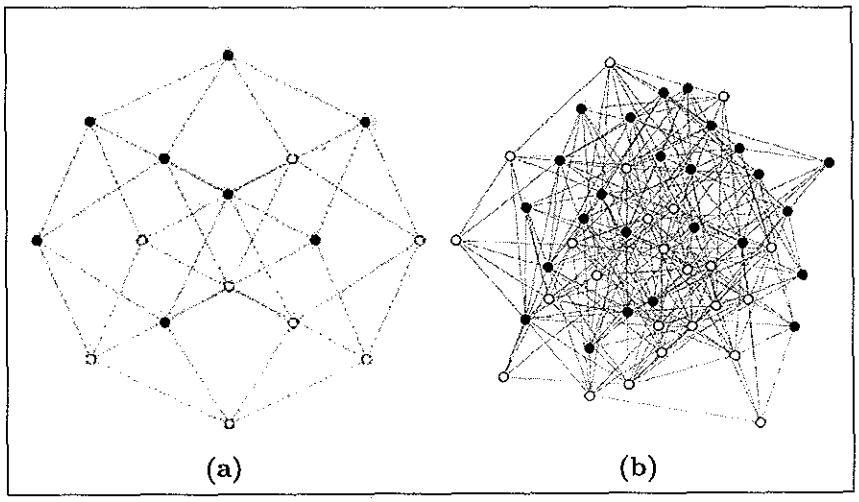


Figura 20: Grafos aleatorios generados con el modelo Watts Strogatz.
a) Grafo con 16 vértices y $p = 0$; b) Grafo con 49 vértices y $p = 0.7$.

IV.2. Análisis y comparaciones de los modelos

Se hizo un análisis empírico con el cual se evaluaron las características más relevantes en los grafos generados con cada modelo, con el objetivo de analizar las diferencias entre cada modelo. Para el análisis se generaron 30 grafos por modelo, tomando en cuenta las características fundamentales de un grafo descritas en Cormen *et al.* (2001). Todos los grafos generados cuentan con 51 vértices y el número de aristas oscila entre $[80, 160]$, es decir se generaron solamente grafos dispersos. En las Tablas II-IV se muestran los promedios de las características para grafos con alrededor de 80, 120 y 160 aristas respectivamente. A continuación se describen las características:

1. Número de aristas: El número de conexiones entre los vértices del grafo.

2. Altura (*height*): En los grafos generados se distingue un vértice raíz. La altura es el número de aristas entre la raíz y el vértice más lejano a ella, donde la noción de distancia es el número minímo de aristas entre un par de vértices.

3. Cintura (*girth*): La cintura o cinturon del grafo es el número de vértices que forman el ciclo más grande en el grafo.

4. Número de clique (*clique number*): Un clique es un conjunto de vértices adyacentes por pares, i.e. un grafo completo es un clique. El número clique de un grafo es el clique más grande en el grafo.

5. Independencia (*independence*): Un conjunto independiente o coclique es un conjunto de vértices de los cuales ningún par es adyacente, es decir no tienen ninguna arista en común.

6. Diámetro (*diameter*): El diámetro de un grafo se puede definir como la distancia mayor de las distancias más cortas entre cualquier par de vértices del grafo.

7. Conectividad de vértices: Es el minímo número de vértices que se tienen que remover para desconectar el grafo; al remover un vértice se remueven sus aristas.

8. Conectividad de aristas: Es el minímo número de aristas que se tienen que remover para desconectar el grafo.

9. Grado total del grafo: El grado de un vértice es el número de aristas que tiene el vértice, el grado total de un grafo es el máximo grado de sus vértices.

10. Grado minímo: Es el grado del vértice con menor número de aristas.

11. Grado máximo: Es el grado del vértice con mayor número de aristas.

Debido a que en los resultados existe una gran similitud entre las características de conectividad de aristas, de vértices y el grado mínimo, solamente se documenta esta última.

Tabla II: Promedio de características de grafos aleatorios (80 aristas).

Característica	Barabasi	Erdos(gnm)	Erdos(gnp)	Evol. E.	Geometr.	Grado A.	Watts S.
Altura	4.4	7.1	6.7	4.9	11.4	5.8	6.2
Cintura	3	3	3	3	3	3.1	3
Número Clique	3	3	3	3	5.7	2.9	3
Independencia	33.5	26.1	25.4	24.7	17.5	20.1	21.9
Diámetro	4.2	7.8	7.4	5.5	12.4	5.6	6.1
Grado Total	3.5	3.1	3.3	3.8	4.5	3.8	3.8
Grado Minímo	1	1	1	1.7	1.2	2.1	1.1
Grado Máxímo	30.3	7.5	7.6	9.7	8.5	4	7.3
Num. Aristas	89.3	80	84.5	95.8	114.3	97.8	93.5

Tabla III: Promedio de características de grafos aleatorios (120 aristas).

Característica	Barabasi	Erdos(gnm)	Erdos(gnp)	Evol. E.	Geometr.	Grado A.	Watts S.
Altura	3.9	5.2	5.4	4.1	10.7	4.6	6.2
Cintura	3	3	3	3	3	3	3
Número Clique	4	3	3.2	3.6	6.3	3.1	3
Independencia	32.2	22	22.4	21.4	16.2	17.3	23.9
Diámetro	3.5	5.3	5.4	4.1	11.2	4	6.4
Grado Total	4.6	4.7	4.5	5.4	5.2	5.6	3.8
Grado Minímo	1.1	1.3	1.3	2.3	1.1	3.9	1
Grado Máxímo	42.2	9.6	9.7	12.7	10.1	6	8
Num. Aristas	117.5	120	116.3	137.3	133.5	144.2	96.5

Tabla IV: Promedio de características de grafos aleatorios (160 aristas).

Característica	Barabasi	Erdos(gnm)	Erdos(gnp)	E. Estoc.	Geometr.	G. Ady.	W. Strog.
Altura	3.5	4.5	4.2	4	7.3	4	4.8
Cintura	3	3	3	3	3	3	3
Número Clique	4.1	3.5	3.5	4.1	6.8	3.2	3.4
Independencia	31.7	19.3	19.4	19.2	14.5	15.7	20.3
Diámetro	2.8	4.1	4.2	4	7.3	3.3	4.6
Grado Total	5.1	6.2	6.2	6.9	6.2	7.3	5.5
Grado Minímo	1.2	1.7	1.9	2.9	1.8	5.3	1.7
Grado Máxímo	47	11.8	11.8	14.9	11.1	8	10.6
Num. Aristas	129.3	160	159.3	177.7	159.4	188.6	144.8

Algunas observaciones interesantes que se obtuvieron al realizar estas comparaciones son las siguientes:

1. Solamente con el modelo de Erdös Rényi se puede tener control en el número de aristas.

2. En todos, excepto el modelo Watts Strogatz, se puede controlar el número de vértices.

3. El modelo Barabási-Albert es el indicado para experimentar con vértices de alto o bajo grado.

4. El método de grado de adyacencia es útil para cuando se busca tener el control del grado de los vértices.

5. Para generar rejillas comunes se puede utilizar el modelo Watts Strogatz, aunque los grafos generados son isomorfos cuando el parámetro $p = 0$.

En los experimentos que se reportan en los Capítulos VI y VII se utilizan estos grafos como entrada del algoritmo genético. Para mayor legibilidad de los resultados se utilizan los acrónimos ilustrados en la Tabla V. Algunas entradas se les inserta un determinado número de aristas para garantizar que la entrada contenga el óptimo, al final del acrónimo de estas entradas se coloca un (*), también se utilizan grafos completos.

Tabla V: Acrónimos de los grafos para la experimentación.

Acrónimo	Descripción
barabasi 1	Grafo generado con el modelo Barabasi-Albert con la raíz con un bajo grado de conectividad.
barabasi 2	Grafo generado con el modelo Barabasi-Albert con la raíz con un alto grado de conectividad.
erdoreny 1	Grafo generado con el modelo Erdös Rényi con la variante $G(n, m)$.
erdoreny 2	Grafo generado con el modelo Erdös Rényi con la variante $G(n, p)$.
evolstoc	Grafo generado con el modelo de Evolución estocastica.
geometri	Grafo generado con el modelo Geométrico.
gradoady	Grafo generado con el modelo Grado de Adyacencia.
watzstro	Grafo generado con el modelo Watts Strogatz.
completo	Grafo completo con todas las $n(n-1)/2$ aristas posibles.

Capítulo V

Algoritmo genético

En la Sección II.3 se da una definición de algoritmo genético así como el esquema general que éste sigue. En este capítulo se explican y describen las partes claves del esquema del algoritmo genético, que son la generación de la población inicial, la selección de padres y la selección de sobrevivientes. Asimismo, se describe el funcionamiento de las distintas representaciones (NetKeys, EdgeSet y dandelion) que se implementan en el algoritmo genético; por cada representación se explican sus operadores de variación y su implementación. Por último, se especifican los tres tipos de algoritmos híbridos que se utilizan, incluyendo el micro algoritmo genético y los métodos de perturbación.

V.1. Generación de individuos para la población inicial

La finalidad de una población en un algoritmo genético es la de almacenar las soluciones candidatas al problema. Una población es un conjunto de genotipos o individuos. Los individuos son objetos estáticos que no cambian, ni se adaptan; es la población la que lo hace. Cuando un algoritmo evolutivo busca una solución en un espacio de árboles de esparcimiento, su población inicial consiste de cromosomas que representan árboles aleatorios. En un algoritmo génetico es deseable tener una población lo suficientemente diversa para que se pueda explorar el espacio de búsqueda de la forma más completa posible. Como se ha mencionado, una solución válida para el PAG es un árbol de es-

parcimiento, por lo que la población es un conjunto de árboles de espacimiento. Raidl y Julstrom (2003) experimentan con tres tipos diferentes de generadores de árboles aleatorios. El primero, *primRST*, utiliza un algoritmo de Prim que se modifica para generar árboles aleatorios de poca altura relativa, es decir, de menor altura que los otros dos métodos. Otro algoritmo, *KruskalRST*, genera árboles aleatorios de mayor altura que *PrimRST*, el cual utiliza un algoritmo de Kruskal; por último, el algoritmo *RandWalkRST* genera árboles de esparcimiento con probabilidad uniforme a través de caminatas aleatorias por el grafo. El principal objetivo de generar árboles aleatorios es obtener soluciones candidatas que conformen la población inicial del algoritmo génetico. Estas soluciones actuan como semilla del proceso evolutivo.

V.1.1. Algoritmos para generar árboles aleatorios

A continuación se describen, en forma de pseudocódigo, los algoritmo propuestos por Raidl y Julstrom (2003) en su trabajo; además, ellos realizan una comparación entre estos métodos. En todos los métodos, las últimas tres letras de los nombres se deben a las siglas en inglés de *Random Spanning Tree*.

PrimRST

El árbol de esparcimiento aleatorio Prim se basa en el algoritmo para encontrar un árbol de esparcimiento minímo de Prim, con la diferencia de que en lugar de seleccionar la arista de menor peso, se selecciona una arista de forma aleatoria. El algoritmo original de Prim, en su versión voraz, construye un árbol de esparcimiento mínimo a partir de un vértice inicial y anexa repetidamente un nuevo vértice que se conecta por medio de la arista con menor peso para así crecer el árbol. Seleccionar cada nueva arista de manera aleatoria en lugar de hacerlo en base a su costo es el procedimiento que se define como

PrimRST. En el Algoritmo 7 se observa un ejemplo de su implementación.

Algoritmo 7: Generador de árbol aleatorio de Prim.

Input : Grafo (V, E)

Output : Árbol de esparcimiento T

```
1  T ← ∅;
   ▷ Selecciona un vértice inicial aleatorio
2  s ← random{ v ∈ V};
   ▷ Conjunto de vértices conectados
3  C ← {s};
   ▷ Aristas elegibles
4  A ← {e ∈ E|e = (s, v), v ∈ V};
5  while C ≠ V do
       ▷ Selecciona una arista de forma aleatoria
6      edge ← random{(u, v) ∈ A|u ∈ C};
7      A ← A − {edge};
8      if v ∉ C then
           ▷ Conecta v al árbol parcial
9          T ← T ∪ (u, v);
10         C ← C ∪ {v};
11         A ← A ∪ {e ∈ E|e = (u, w)w ∉ C};
12     end
13 end
14 return T
```

Sea el grafo $G = (V, E)$ con $|V| = n$ y $|E| = m$. Si G se representa por medio de una lista de adyacencias, se puede hacer una identificación rápida de las aristas adyacentes a cada nuevo vértice conectado y la implementación de PrimRST se ejecuta en tiempo $O(m)$. Si G es completo, solo es necesario mantener el registro del conjunto de vértices actualmente en el árbol de esparcimiento (conjunto C en el Algoritmo 7) y su complemento $(V - C)$; cada nueva arista se identifica al seleccionar aleatoriamente un vértice de este último conjunto, por lo que el espacio y tiempo del algoritmo es $O(n)$ para ambos. PrimRST es simple, eficiente y además se puede extender fácilmente para que regrese árboles con distintos requerimientos. Desafortunadamente, PrimRST obtiene árboles de cierta estructura con más alta probabilidad que otros. Las aristas adyacentes del vértice inicial que están en el conjunto A son las aristas elegibles desde el principio, por lo que son las que tienen más posibilidades de incluirse en el árbol que otras aristas. Así que, los árboles que tienen mayor probabilidad bajo PrimRST son los árboles en forma de estrella enraizada. Una explicación más extendida de lo anterior se hace en el trabajo de Raidl y Julstrom (2003).

KruskalRST

El algoritmo de Kruskal es una estrategia voraz para construir un árbol de esparcimiento minímo de un grafo G. Este algoritmo examina las aristas de G por orden incremental de su peso e incluye en el árbol aquella que conecte componentes previamente desconectados. El método de KruskalRST elige las aristas de forma aleatoria en lugar de por orden incremental; este método se ilustra en el Algoritmo 8. El tiempo total del algoritmo es $O(m)$. KruskalRST genera con más probabilidad cierto tipo de árboles que otros, un análisis a profundidad de este comportamiento se encuentra en la investigación original (Raidl y Julstrom, 2003).

Algoritmo 8: Generador de árbol aleatorio de Kruskal.

Input : Grafo (V, E)

Output : Árbol de esparcimiento T

```
1  T ← ∅;
   ▷ Se duplica el conjunto de aristas
2  A ← {E};
3  while |T| < |V| − 1 do
       ▷ Selecciona una arista de forma aleatoria
4      (u, v) ← random{(u, v) ∈ A};
5      A ← A − {(u, v)};
       ▷ Si u y v no están conectados en T
6      if (u and v ∉ T) then
7          T ← T ∪ (u, v);
8      end
9  end
10 return T
```

RandWalkST

Los dos métodos anteriores, PrimRST y KruskalRST, asocian probabilidades no uniformes a los árboles de esparcimiento de un grafo G, aunque este último es menos sesgado que el primero. Broder (1989) describe un método probabilístico basado en una caminata aleatoria en un grafo G. La caminata comienza en un vértice arbitrario de G, a cada paso se mueve sobre una arista adyacente que se selecciona de forma aleatoria de uno de sus vecinos. Cuando se visita por primera vez un vértice, su arista conecta

el árbol de esparcimiento. El algoritmo termina cuando se visitan todos los vértices, por lo que se obtiene un árbol de esparcimiento válido de G. Este método es el que se muestra en el Algoritmo 9 y lleva por nombre RandWalkRST.

Algoritmo 9: Generador de árbol aleatorio RandWalk.

Input : Grafo (V, E)

Output : Árbol de esparcimiento T

```
1  T ← ∅;
   ▷ Selecciona un vértice aleatorio v_0 y lo marca como visitado
2  v_0 ← random{ v ∈ V};
3  v_0.visited ← true;
4  while |T| < n − 1 do
       ▷ Selecciona un vecino aleatorio de v_0
5      v_j ← {random v_j ∈ V : (v_j, v_0) ∈ E};
       ▷ Si v_j no ha sido visitado
6      if (v_j.visited == false) then
7          T ← T ∪ (v_0, v_j);
           ▷ Marcar v_j como visitado
8          v_j.visited ← true;
9      end
10     v_0 ← v_j
11 end
12 return T
```

En Broder (1989) se demuestra que este proceso regresa un árbol de esparcimiento con probabilidad uniforme. El tiempo de ejecución esperado de este algoritmo es

$O(n\ log\ n)$ para casi todos los grafos y $O(n^3)$ para algunos casos especiales.

V.1.2. Análisis y comparaciones

Raidl y Julstrom (2003) examinan las propiedades de los árboles de esparcimiento que se generan con PrimRST, KruskalRST y RandWalkRST para grafos completos de diferentes tamaños y miden los diámetros de los árboles de esparcimiento resultantes. El diámetro de un árbol, de acuerdo a la definición de la Sección IV.2, es el número de aristas en la trayectoria más larga en el árbol. El diámetro para un árbol con forma de estrella enraizada es de dos, y en un árbol con forma de cadena lineal es de $n-1$. Los resultados que obtienen Raidl y Julstrom (2003) son que PrimRST genera árboles con un diámetro promedio menor que los otros dos métodos. KruskalRST genera árboles de diámetro de promedio más grande que los de PrimRST. Finalmente con RandWalkRST se obtienen los árboles con mayor diámetro promedio cuando se compara con los dos anteriores. Estas diferencias son consistentes e incrementales conforme a n.

En esta tesis, estos tres métodos se implementan y comparan entre sí, para generar tablas comparativas del promedio de 30 árboles por método; además, que a diferencia de Raidl y Julstrom (2003), que solo usan grafos completos en esta comparación, se utilizan como entradas grafos aleatorios de los modelos que se describen en la Sección IV.1. Los árboles que se obtienen se evalúan tomando en cuenta las características de altura, diámetro, grado máximo y número independiente de los árboles. Las Tablas VI-VIII muestran estas comparaciones. En la Tabla IX se muestran los promedios para todos los grafos aleatorios de los métodos para árboles aleatorios.

Con base en las tablas, se puede observar que el método que genera árboles con mayor diámetro y altura es RandWalkRST, seguido por KruskalRST y el método con

Tabla VI: Comparación de las caracterísiticas de PrimRST.

Característica	Barabasi	Erdos(gnm)	Erdos(gnp)	Evol. E.	Geometr.	Grado A.	Watts S.
Altura	7.4	10.1	10.6	9.9	14.4	10.9	10.9
Independiente	38.1	29.9	30.4	30.7	29.4	29.4	29.5
Diámetro	8.2	12.1	12.3	12.1	16.7	12.4	12.5
Grado Máxímo	22.9	5.6	6.1	6.1	5.1	5.5	5.5

Tabla VII: Comparación de las caracterísiticas de KruskalRST.

Característica	Barabasi	Erdos(gnm)	Erdos(gnp)	Evol. E.	Geometr.	Grado A.	Watts S.
Altura	8.6	13.9	13.1	13.5	18.1	15.3	14.1
Independiente	37.8	29.3	29.5	29.5	28.6	28.8	29.2
Diámetro	10.2	17.4	16.0	16.2	20.6	18.5	17.1
Grado Máxímo	19.5	4.9	5.0	5.3	4.6	4.6	5.1

Tabla VIII: Comparación de las caracterísiticas de RandWalkRST.

Característica	Barabasi	Erdos(gnm)	Erdos(gnp)	Evol. E.	Geometr.	Grado A.	Watts S.
Altura	9.8	15.9	14.9	14.7	18.3	16.7	15.7
Independiente	36.4	29.1	29.1	29.1	29	28.6	29.3
Diámetro	11.2	18.3	18.9	18.9	21.8	20.1	18.8
Grado Máxímo	18.6	4.9	4.7	4.6	4.4	4.4	4.9

Tabla IX: Comparación de las caracterísiticas promedio de PrimRST, KruskalRST y RandWalkRST.

Característica	KruskalRST	PrimRST	RandWalkRST
Altura	13.8	10.6	15.9
Independiente	30.4	31.0	30.1
Diámetro	16.6	12.3	18.3
Grado Máxímo	7.0	8.1	6.6

árboles de menor altura es PrimRST. Con esto se reafirma el análisis que realizó Raidl y Julstrom (2003). En cuanto al tamaño del conjunto independiente y el grado máximo, la diferencia no es significativa entre los distintos métodos. Es necesario señalar que dados los resultados anteriores, los métodos generan árboles distintos y ésto es algo deseable para la generación inicial de la población del algoritmo genético.

V.2. Métodos de selección de padres y selección de sobrevivientes

Hasta el momento, al hablar de algoritmos genéticos, solo se ha hecho hincapié en el mapeo de los individuos, en cómo generar población de individuos diversos, y que los operadores de variación trabajan en estos individuos para producir descendencia. Esta descendencia generalmente hereda algunas de las propiedades de los padres, pero a su vez difieren un poco de ellos, proporcionando nuevas soluciones potenciales a evaluar. Ahora se discuten un par de elementos importantes en el proceso evolutivo, los cuales se encargan de diferenciar a los individuos con base en su aptitud para competir por los recursos (sobrevivientes) y para participar en la fase de reproducción.

Existen dos diferentes modelos de algoritmos genéticos en la literatura: el modelo generacional y el modelo de estado estacionario; para definir estos modelos, primero es necesario mencionar lo siguiente: en cada generación se comienza con una población de tamaño μ, de la que se selecciona un conjunto λ de individuos a aparearse o padres. La descendencia se crea a partir de los padres al aplicar los operadores de variación.

Si el tamaño de la descendencia es igual al tamaño de la población, es decir, si $\mu = \lambda$ y la población original se reemplaza por su descendencia se le llama modelo generacional.

En el modelo de estado estacionario sólo se reemplaza una parte de la población, para este caso $\mu > \lambda$. El porcentaje de la población que se reemplaza se denomina brecha generacional y es igual a λ/μ. Para el algoritmo genético se toma una combinación de ambos modelos, siempre se genera un tamaño de descendencia igual al tamaño de la población ($\mu = \lambda$), pero no necesariamente ocurre un reemplazo generacional, si no que los descendientes pueden reemplazar solo una parte de la población original, dependiendo del criterio de selección.

Como se vió en la descripción general de un algoritmo genético en la Sección II.3, hay dos puntos en el ciclo evolutivo en el que la competencia con base en la aptitud puede ocurrir, durante la selección de padres y durante la selección de sobrevivientes, los que a continuación se presentan.

V.2.1. Selección de padres

El objetivo de la selección de padres es resaltar a los individuos basándose en la medida de su aptitud, en particular, permite que los mejores individuos se vuelvan padres de la siguiente generación. Un individuo es un padre si se elije para realizar algún proceso de variación con la finalidad de crear descendencia. Los mecanismos de selección de sobrevivientes y selección de padres son los responsables de la mejora en la calidad de la población. En cómputo evolutivo, la selección de padres es por lo general probabilística, de tal manera que los mejores individuos tienen una mayor probabilidad de convertirse en padres que aquellos que tienen una aptitud baja en comparación del resto de la población. Sin embargo, regularmente a los individuos con baja aptitud se les da una pequeña probabilidad; de otra forma toda la búsqueda puede volverse completamente voraz y quedar atrapada en un óptimo local.

Durante esta investigación se evalúan distintos criterios de selección de padres, se implementan algunos de los métodos de selección que más se utilizan y otros propuestos, con el fin de definir el mejor método de selección para el algoritmo genético. Se utiliza una alberca de apareamiento (*mating pool*), con la que se almacena el conjunto de individuos a aparearse, su tamaño está dado por $\lambda = P_c * \mu$, donde P_c es la probabilidad de cruzamiento y μ el tamaño de la población original, esta alberca de apareamiento se llena de acuerdo a los siguientes criterios de selección de padres:

Probabilidad de recombinación aleatoria

En el criterio de selección por probabilidad de recombinación aleatoria (*random crossover rate*) se ordena la población de tamaño (μ) en relación a su aptitud y se van tomando individuos de dos en dos, este par de individuos se incluyen en la alberca de apareamiento con un probabilidad de P_c, sin tomar en cuenta el tamaño λ.

Probabilidad uniforme

En la selección de padres por probabilidad uniforme, todo el conjunto (μ) de individuos tienen la misma probabilidad de elegirse ($1/\mu$), éstos se van almacenando de uno en uno hasta tener el tamaño de la alberca de apareamiento que se requiere. Se puede destacar que en este caso se asegura de no tener un conjunto de padres sesgado; sin embargo, no asegura mejorar la calidad de la población, dejando toda la responsabilidad al proceso de selección de sobrevivientes.

Elitista

Para el criterio elitista, se ordena la población de tamaño μ dependiendo de su aptitud y se eligen los mejores λ individuos para que se crucen entre sí. Nótese que este

criterio puede llevar a una convergencia prematura, es decir, a quedarse atrapado en un óptimo local, pero asegura mejorar la calidad de la población.

Elitismo parcial

El criterio de elitismo parcial es una combinación de los dos anteriores, ya que selecciona la mitad de los individuos de forma elitista y la otra mitad de forma uniforme. Esta combinación tiene como objetivo tener los aspectos buenos del criterio elitista y del criterio uniforme.

Torneo

El torneo es un método de selección de padres bastante usado en la literatura, que también se le llama *tournament-k*. Para este criterio se eligen k individuos de la población de tamaño μ, éstos compiten entre sí y se obtiene el mejor individuo basándose en su aptitud. Este individuo se convierte en un padre para la próxima generación al incluirlo en la alberca de apareamiento. Este criterio depende mucho del tamaño de k, ya que entre más pequeño, el tiempo es menor, pero tiende a seleccionar individuos de menor aptitud; por otro lado, entre más grande el torneo, mayor será la probabilidad de seleccionar un buen individuo, pero a su vez mayor será el tiempo de ejecución.

Ruleta

Para el criterio de selección por ruleta (*roulette wheel*) los individuos tienen una probabilidad de elegirse proporcional a su aptitud. La probabilidad de que un individuo i con aptitud (*fitness*) f_i se seleccione para reproducirse es $f_i / \sum_{j=1}^{\mu} f_j$, esto es que la probabilidad de selección depende de la aptitud del individuo cuando se compara con los valores de aptitud del resto de la población.

Los tipos de criterios de selección de padres que se mencionan con anterioridad se compararon por medio de algunas pruebas preliminares. En los resultados no se encontró diferencia significativa en cuanto a las soluciones obtenidas entre uno y otro criterio; lo único destacable es que el criterio por torneo con $k = 10$ para una población de 200 individuos fue la que más tiempo tardó con respecto a las otras.

V.2.2. Selección de sobrevivientes

La finalidad de la selección de sobrevivientes es la de destacar entre los individuos a aquellos con buena aptitud. En esto es similiar a la selección de padres; sin embargo, la selección de sobrevivientes se utiliza en una etapa distinta del ciclo evolutivo. El mecanismo de selección de sobrevivientes se utiliza después de generar la descendencia por medio de los operadores de variación (recombinación y mutación). En los algoritmos genéticos, el tamaño de la población es en general constante, entonces la decisión que se tiene que tomar es cuáles individuos serán los sobrevivientes para la próxima generación. Esta decisión, por lo general, se basa en los valores de aptitud de los individuos, el concepto de edad o antigüedad de un individuo se utiliza frecuentemente. Al contrario de la selección de padres, que suele ser estocástica, la selección de sobrevivientes es principalmente determinística, por ejemplo, ordenando los padres y descendientes por su aptitud (tendencia por aptitud) o seleccionando solo individuos de los descendencientes (tendencia por edad).

Al igual que en la selección de padres, en la selección de sobrevivientes se implementan varios criterios de selección que se describen a continuación:

Basado en aptitud

El criterio basado en aptitud (*fitness based*) ordena los $(\mu+\lambda)$ individuos de acuerdo a su aptitud, para después seleccionar solamente los mejores μ individuos. Se puede observar que este criterio es totalmente elitista.

Aleatorio

El criterio aleatorio selecciona de forma aleatoria μ individuos de la población total $(\mu + \lambda)$; al contrario del criterio anterior, éste no utiliza ninguna clase de elitismo por ser totalmente aleatorio.

Mitad padres y mitad hijos

En este criterio se ordenan los conjuntos de descendientes (λ) y la población original (μ) de acuerdo a su aptitud y se eligen como sobrevivientes los mejores $(\mu/2)$ individuos de cada conjunto. Con este criterio se mantienen los mejores individuos de la población original y los mejores descendientes.

Mejores distintos

Este criterio es muy similar al criterio basado en aptitud, con la diferencia que se eliminan los individuos que son iguales en aptitud y en estructura del árbol, es decir, individuos que representan la misma solución (para determinar esto se utiliza la distancia de Hamming). Para el problema en cuestión se define la distancia entre dos árboles T_1 y T_2 como el número de aristas que pertenecen a T_1 pero no a T_2 (o de forma equivalente, el número de aristas que se requieren intercambiar para tranformar T_1 a T_2). Por lo tanto, la distancia entre dos árboles distintos en T_n es siempre un entero en el intervalo $[1, n-1]$. Dos árboles están separados por una distancia $\Delta \in [1, n-1]$, si

se requieren Δ aristas para llegar de un árbol a otro (Paulden y Smith, 2006a), esta distancia es simétrica.

En la experimentación se utiliza el criterio basado en aptitud y sólo para el micro algoritmo genético que se explica en la Sección V.6.2, se utiliza el criterio de los mejores árboles distintos.

V.3. Funcionamiento de network random keys

La representación network random keys (NetKeys), así como su mapeo a un árbol de esparcimiento, se describe en la Sección II.4.2. En esta sección se explican los detalles de su implementación en el algoritmo genético, se describe la generación de la población inicial, así como los operadores evolutivos que se utilizan con esta representación.

V.3.1. Población inicial

NetKeys es la única representación que se implementa en esta tesis con la que no se utilizan los métodos para generar árboles aleatorios descritos en la Sección V.1.1. Para generar la población inicial, la representación NetKeys trabaja con todas las aristas del grafo de entrada, ya que para la creación de un individuo de la población inicial es necesario asignar pesos aleatorios a todas las aristas del grafo de entrada; después se genera un árbol de esparcimiento máximo tomando en cuenta los pesos de las aristas; este proceso se repite hasta que se tenga el número de individuos μ que se requiere para la población. El proceso del mapeo se precisa a profundidad en la Sección II.4.2. Nótese que NetKeys es la única representación de las se implementa que en su genotipo representa todas las aristas del grafo de entrada. En la Figura 21 se observa un ejemplo del proceso de inicialización, dado un grafo de entrada de 5 vértices, se le asignan pesos

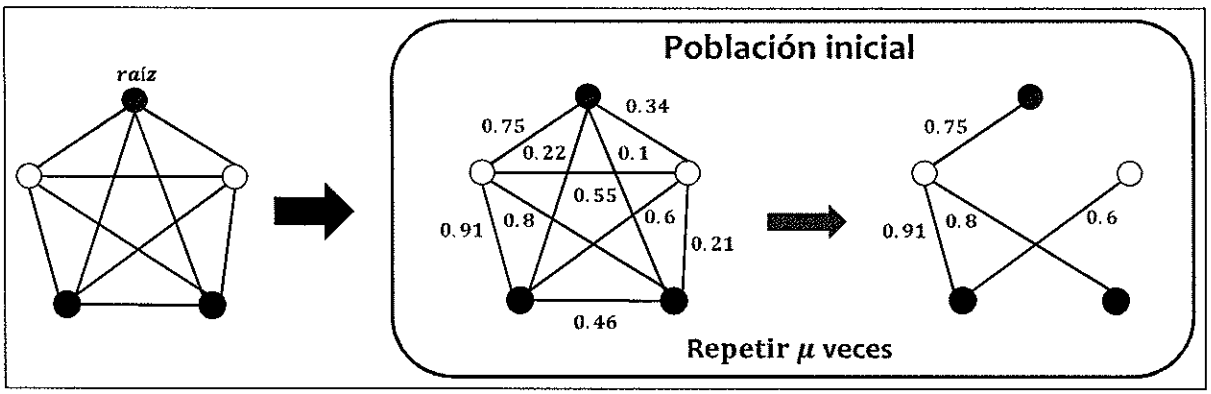


Figura 21: Población inicial para la representación NetKeys.

aleatorios a las 10 aristas en el grafo y se obtiene el árbol de esparcimiento máximo; el proceso de asignar pesos aleatorios y obtener el árbol de esparcimiento máximo se repite μ veces.

V.3.2. Operadores evolutivos

Los operadores evolutivos en un algoritmo genético son la recombinación y la mutación de individuos. En el caso de la recombinación, dado que es un operador binario, se seleccionan dos individuos o padres para obtener dos descendientes o hijos que hereden algunas de las características de los padres. El caso de la mutación es un operador unario ya que se toma solo un individuo, para después obtener un nuevo individuo por medio de la perturbación del individuo original. NetKeys es una representación de punto flotante, es decir que los alelos en el genotipo son en realidad números reales. Se requiere usar un operador de recombinación, que por cada posición en el nuevo genotipo, se cree un nuevo alelo cuyo valor se encuentre entre el de los padres. De manera formal, se puede decir que si se crea un descendiente z de los padres x e y, entonces el alelo para el gen i es dado por $z_i = \alpha x_i + (1-\alpha)y_i$ para algun $\alpha \in [0,1]$. Los operadores de este tipo se conocen como de recombinación aritmética (*arithmetic recombination*).

El parámetro α es a veces aleatorio, pero en la práctica el uso más común es un valor constante, usualmente 0.5 (*uniform arithmetic recombination*). Nótese que con una recombinación aritmética es posible generar nueva información a los genes; lo anterior tiene la desventaja que cada vez que se promedia el valor de los genes de ambos padres se reduce el intervalo de valores que puede tomar el gen. Eiben y Smith (2003) describen tres tipos de recombinación aritmética.

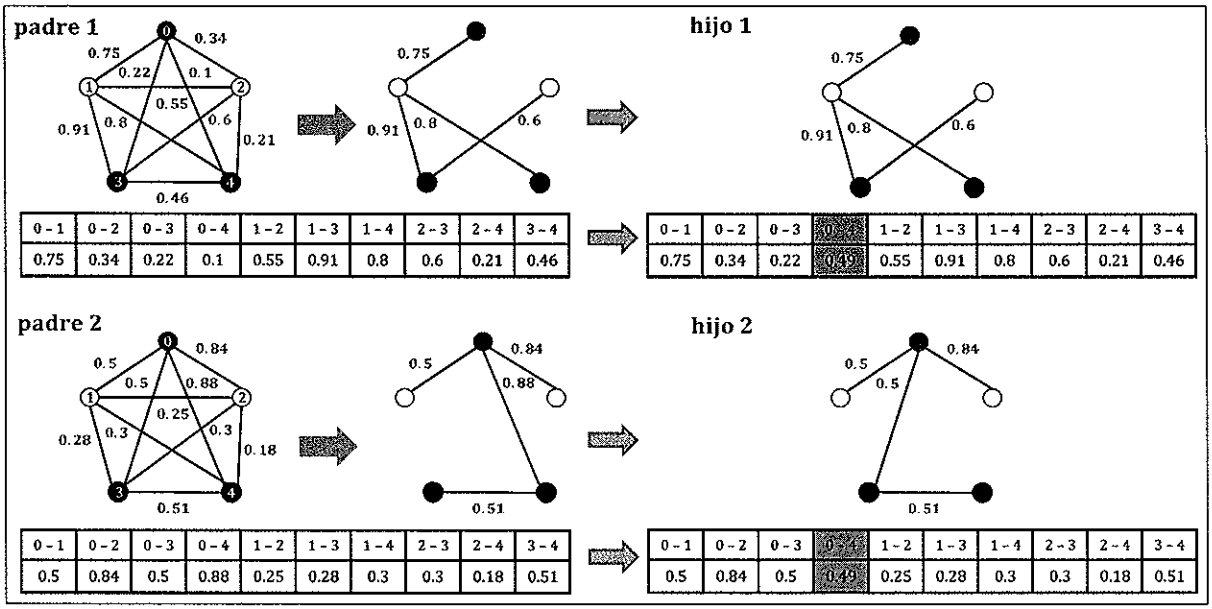
Para el caso del operador de mutación se utiliza un operador simple de cambio de peso. Los tres operadores de recombinación y el operador de mutación se implementan para el algoritmo genético y se describen a continuación:

Recombinación aritmética sencilla

Para este operador se selecciona un punto k de recombinación, en este punto se hace un promedio aritmético de ambos padres. Los otros puntos se copian exactamente iguales a los del padre 1, para el hijo 1, y del padre 2, para el hijo 2.

$$hijo\ 1 : \langle x_1, ..., x_{k-1}, \alpha \cdot y_k + (1-\alpha) \cdot x_k, x_{k+1}, ..., x_n \rangle.$$
$$hijo\ 2 : \langle y_1, ..., y_{k-1}, \alpha \cdot x_k + (1-\alpha) \cdot y_k, y_{k+1}, ..., y_n \rangle.$$

Por último, después de tener los genotipos modificados de los descendientes, simplemente se obtiene el árbol de esparcimiento máximo de cada uno. En la Figura 22 se ve un ejemplo del operador de recombinación aritmético sencillo, para este ejemplo en particular $k = 3$ y $\alpha = 0.5$, por lo que $z_3 = 0.5(0.1) + 0.5(0.88) = 0.49$. En la Figura 22 se puede observar el grafo de entrada con pesos aleatorios asignados, así como el genotipo en forma de arreglo que lo representa, cada posición del arreglo contiene el peso asignado a la arista. Los alelos de los descendientes que están sombreados son los promediados.



0-1	0-2	0-3	0-4	1-2	1-3	1-4	2-3	2-4	3-4
0.75	0.34	0.22	0.1	0.55	0.91	0.8	0.6	0.21	0.46

0-1	0-2	0-3	0-4	1-2	1-3	1-4	2-3	2-4	3-4
0.75	0.34	0.22	0.46	0.55	0.91	0.8	0.6	0.21	0.46

0-1	0-2	0-3	0-4	1-2	1-3	1-4	2-3	2-4	3-4
0.5	0.84	0.5	0.88	0.25	0.28	0.3	0.3	0.18	0.51

0-1	0-2	0-3	0-4	1-2	1-3	1-4	2-3	2-4	3-4
0.5	0.84	0.5	0.49	0.25	0.28	0.3	0.3	0.18	0.51

Figura 22: Recombinación aritmética sencilla.

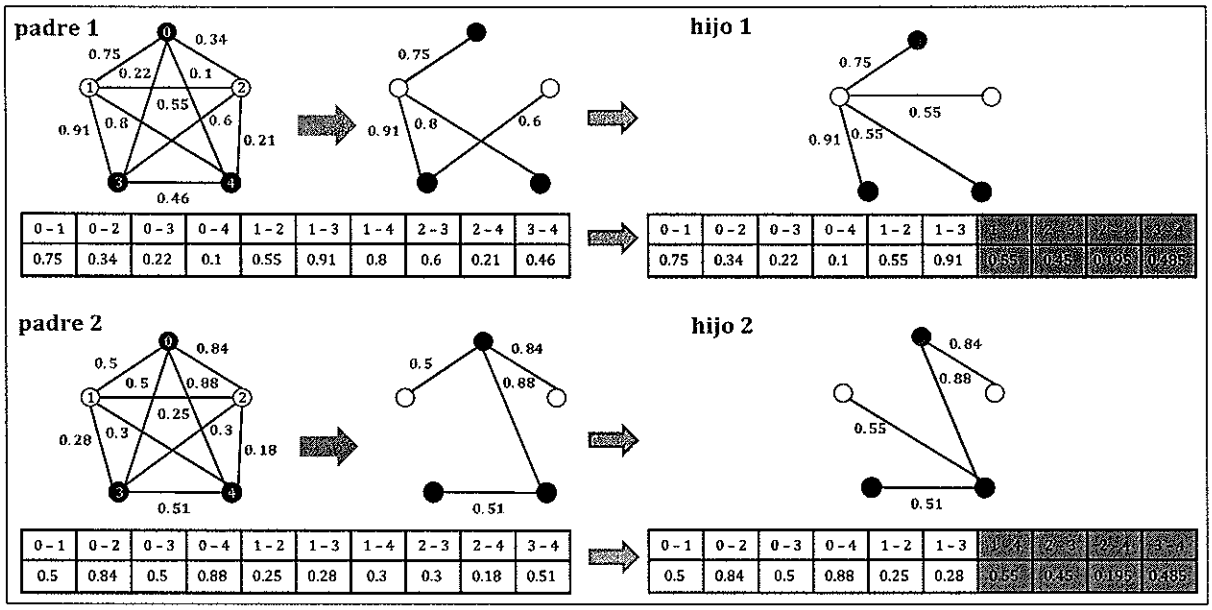
Recombinación aritmética simple

En este operador, al igual que el anterior, se selecciona un punto k de recombinación; después, para el hijo 1 se toman los primeros k alelos del padre 1 y se copian en este hijo. El resto del genotipo es un promedio aritmético entre el padre 1 y el 2, mientras que el hijo 2 se hace de forma análoga, con los padres x e y intercambiados.

$$hijo\ 1 : \langle x_1, ..., x_k, \alpha \cdot y_{k+1} + (1-\alpha) \cdot x_{k+1}, ..., \alpha \cdot y_n + (1-\alpha) \cdot x_n \rangle.$$

En la Figura 23 se ve un ejemplo de este operador de recombinación, para este ejemplo $k = 5$ y $\alpha = 0.5$ y se realizan las siguientes operaciones:

$$z_6 = 0.5(0.8) + 0.5(0.3) = 0.55,\ z_7 = 0.5(0.6) + 0.5(0.3) = 0.45,$$
$$z_8 = 0.5(0.21) + 0.5(0.18) = 0.195,\ z_9 = 0.5(0.46) + 0.5(0.51) = 0.485$$



0-1	0-2	0-3	0-4	1-2	1-3	1-4	2-3	2-4	3-4
0.75	0.34	0.22	0.1	0.55	0.91	0.8	0.6	0.21	0.46

0-1	0-2	0-3	0-4	1-2	1-3	1-4	2-3	2-4	3-4
0.75	0.34	0.22	0.1	0.55	0.91	0.55	0.45	0.195	0.485

0-1	0-2	0-3	0-4	1-2	1-3	1-4	2-3	2-4	3-4
0.5	0.84	0.5	0.88	0.25	0.28	0.3	0.3	0.18	0.51

0-1	0-2	0-3	0-4	1-2	1-3	1-4	2-3	2-4	3-4
0.5	0.84	0.5	0.88	0.25	0.28	0.55	0.45	0.195	0.485

Figura 23: Recombinación aritmética simple.

Recombinación aritmética completa

El operador de recombinación aritmética completa trabaja al tomar la suma de los pesos promediados de los alelos de los padres y copiar el resultado para cada gen en los descendientes, i.e:

$$\text{hijo } 1 : z_i = \alpha \cdot x_i + (1 - \alpha) \cdot y_i$$

$$\text{hijo } 2 : z_i = \alpha \cdot y_i + (1 - \alpha) \cdot x_i$$

Este procedimiento se ilustra en la Figura 24. Como en el ejemplo se utiliza $\alpha = 0.5$, los dos descendientes son dos individuos idénticos para este operador. De los operadores aritméticos, éste es el que se utiliza más comunmente.

Mutación por cambio de peso

La mutación por cambio de peso (*mutation change weight*) de la representación NetKeys se refiere a tomar un alelo aleatorio k del individuo y cambiarlo por un valor

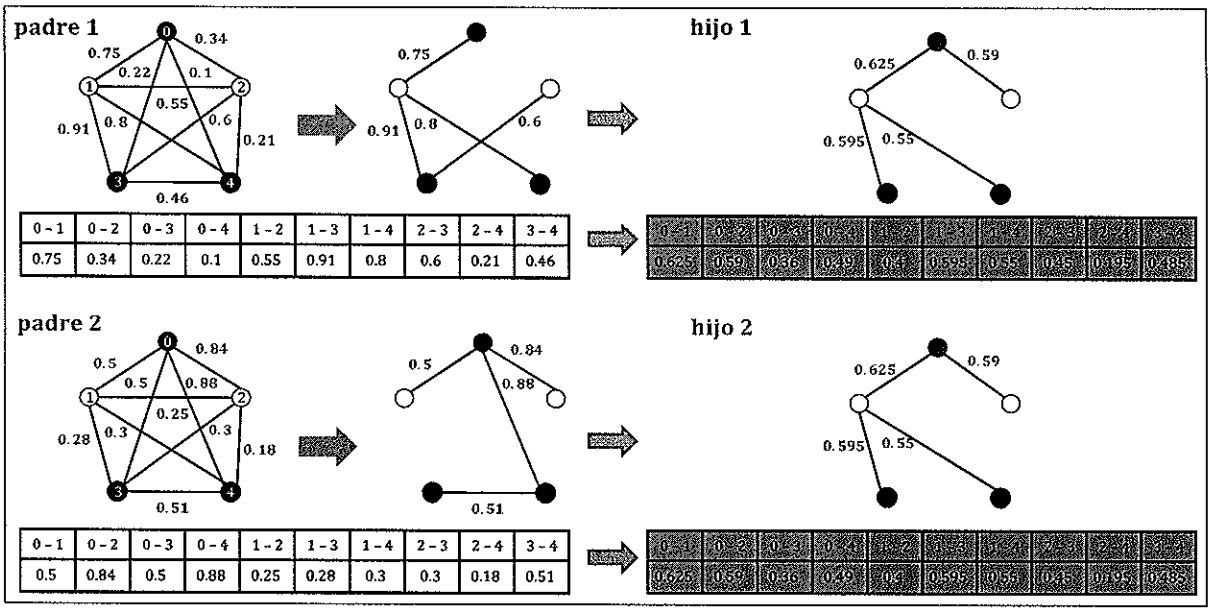


Figura 24: Recombinación aritmética completa.

aleatorio $\delta \in (0, 1)$; se requiere aplicar varias veces este operador, ya que al cambiar un elemento del genotipo original, no se asegura que se efectue un cambio en el árbol que representa. Lo anterior se debe a que en el genotipo se encuentran todas las aristas del grafo de entrada y puede darse el caso que la arista que se cambia no afecte al procedimiento de obtener el árbol de esparcimiento máximo. Es por ello que este operador de mutación se aplica un número determinado de veces.

En el Capítulo VI se realizan algunas comparaciones entre los operadores de recombinación aritmética y el número de veces a aplicar la mutación de cambio de peso para definir qué configuración es la mejor para el algoritmo genético con la representación NetKeys.

V.4. Funcionamiento de EdgeSet

Así como se describe en la Sección II.4.2, la representación EdgeSet es la más directa de todas. EdgeSet consiste en un arreglo con todas las aristas que representan el árbol de esparcimiento. En esta sección se describe el funcionamiento de esta representación en el algoritmo genético desde la generación de la población inicial hasta los operadores evolutivos que se utilizan.

V.4.1. Población inicial

En esta representación se utilizan los generadores de árboles de esparcimiento aleatorios vistos en la Sección V.1.1 para obtener la población inicial. Este proceso se ilustra en la Figura 25. Cada uno de los métodos para generar árboles aleatorios se utiliza para crear una tercera parte de la población inicial, ya que como se vió en la Sección V.1.1, estos métodos tienden a generar árboles de distintas características (lo cual es algo deseable para la población ya que se busca tener diversidad en la población). Nótese que para esta representación no es necesario realizar un proceso de mapeo adicional más que el generar los árboles de esparcimiento aleatorios.

V.4.2. Operadores evolutivos

El objetivo de los operadores de variación es explorar el espacio de búsqueda. En el caso de la recombinación, se combinan dos individuos y se generan dos descendientes que sean similares a sus padres. Mientras que la mutación es útil para perturbar una solución y encontrar nuevas soluciones. Los operadores evolutivos que se implementan para EdgeSet son los que se propusieron en Raidl y Julstrom (2003), ya que demuestran que tienen buenos resultados para distintos problemas (como se ve en la Sección III.4).

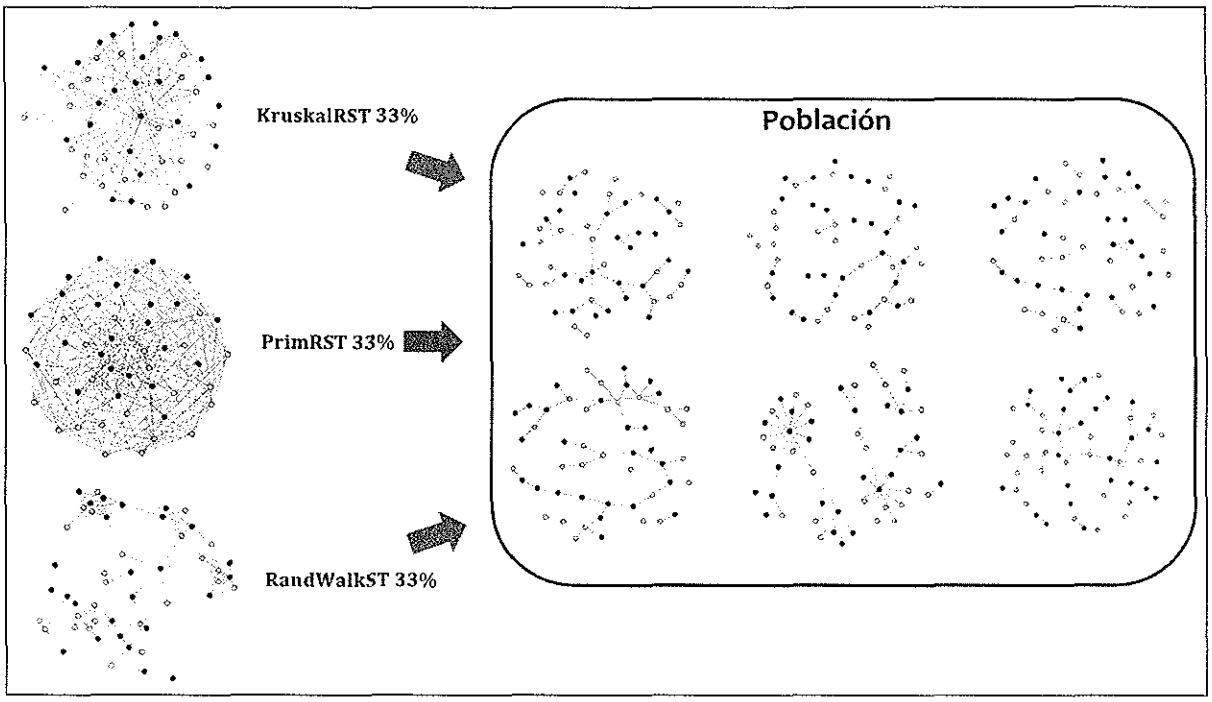


Figura 25: Población inicial en EdgeSet.

Recombinación por unión de árboles

La recombinación por unión de árboles toma las aristas de ambos padres y hace una unión de ellas generando de esta forma un grafo; a partir de este grafo se obtienen dos árboles de esparcimiento aleatorios con los métodos descritos en la Sección V.1.1. De esta forma se obtienen los descendientes que heredan las características o, como en este caso, las aristas de los padres. En la Figura 26 se muestra un ejemplo de esta recombinación.

Mutación por borrado aleatorio de arista

La mutación por borrado aleatorio de arista (*delete random edge*), visto de una forma general, elimina una arista seleccionada de forma aleatoria del árbol original y la reemplaza con otra arista aleatoria que reconecte el árbol. Sin embargo, es necesario

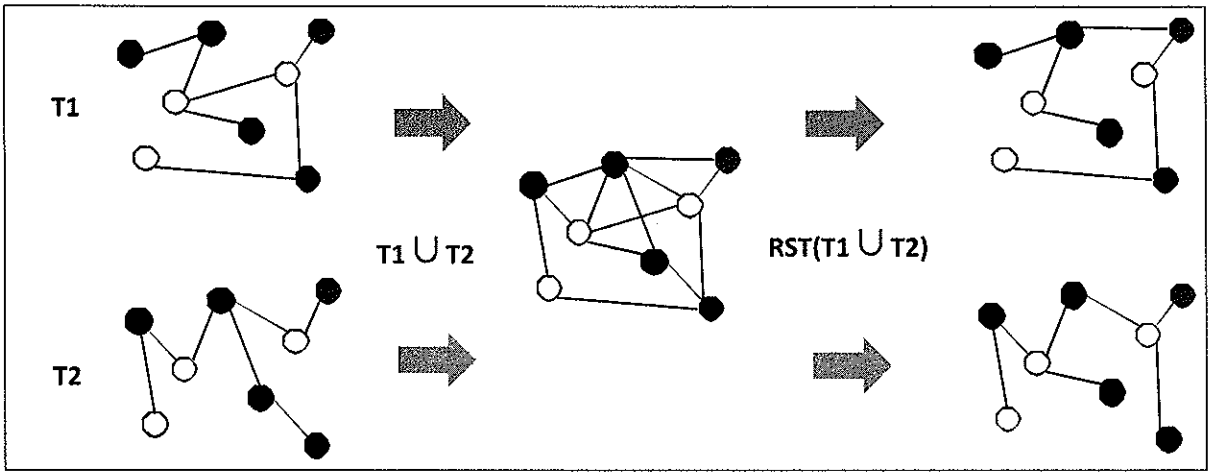


Figura 26: Recombinación por unión de árboles en EdgeSet.

describirlo con un poco más de detalle para comprender su funcionamiento. La mutación por borrado de arista consiste en remover una arista de forma aleatoria del individuo a mutar y dado que el individuo es un árbol, esta acción genera dos componentes desconectados. Para identificar los vértices pertenecientes a cada componente desconectado se pueden realizar dos búsquedas por anchura (Cormen *et al.*, 2001), dando como vértices fuente los extremos de la arista que se elimina.

Para unir de nuevo estos componentes se selecciona una arista de forma aleatoria de las aristas que conecte nuevamente los componentes y genere un árbol de esparcimiento. Se puede notar que la distancia de Hamming entre el individuo original y el mutado es siempre a lo más uno. Este operador puede implementarse con estructuras de datos de conjuntos disjuntos (Cormen *et al.*, 2001). Este operador de mutación se muestra en la Figura 27.

Mutación por adición aleatoria de arista

La mutación por adición aleatoria de arista (*add random edge*) puede ser muy similar a la mutación por borrado aleatorio de arista, dependiendo de su implementación,

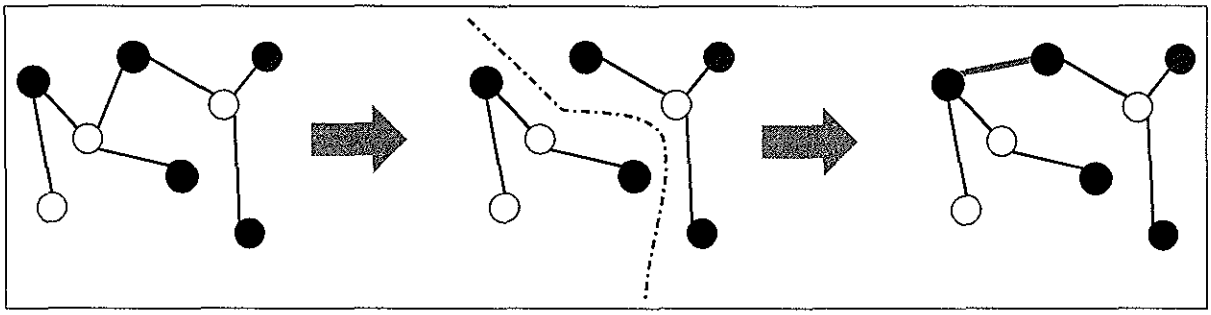


Figura 27: Mutación de borrado aleatorio de arista en EdgeSet.

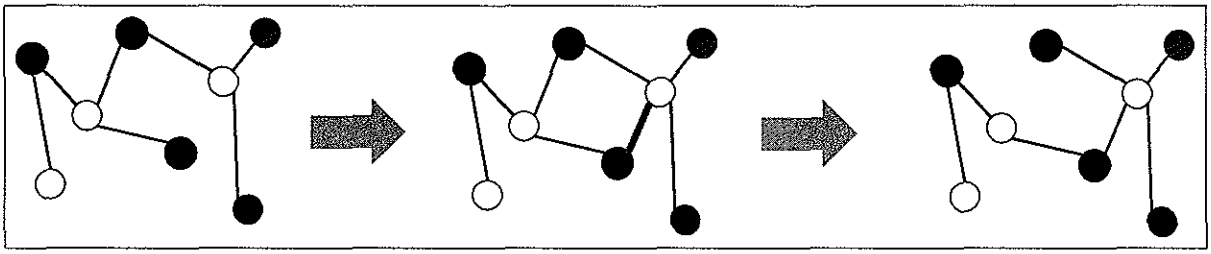


Figura 28: Mutación de adición aleatoria de arista en EdgeSet.

ya que dado el mismo árbol para ambos operadores, pueden generar el mismo rango de individuos con la misma probabilidad. La forma en que trabaja este operador es agregando de forma aleatoria una arista del grafo de entrada, siempre y cuando esta arista no exista en el árbol original; como consecuencia el individuo ya no es un árbol si no que es un grafo con un ciclo. El siguiente paso es identificar las aristas que pertenecen al ciclo y remover una de ellas de manera aleatoria. Este último paso puede ser un poco más costoso dependiendo de la implementación. La Figura 28 muestra un ejemplo de esta mutación.

V.5. Funcionamiento de Dandelion

Dandelion es la representación con el mapeo más complejo de las tres representaciones que se implementan. Este mapeo se describe a detalle en la Sección II.4.2. En esta sección se explican los operadores de variación para dandelion. Cabe señalar que ésta

es la única representación para la que no se hizo una comparación empírica de sus operadores de variación, es decir que se utilizan los que Paulden y Smith (2006b) reportan con buenos resultados; por lo que no se incluyen comparaciones en el Capítulo VI de experimentos preliminares.

V.5.1. Población inicial

De manera similar que para EdgeSet, en dandelion se utilizan los generadores de árboles de esparcimiento aleatorios vistos en la Sección V.1.1 para generar los individuos de la población inicial. Después se realiza el mapeo de cada árbol a su representación dandelion. En la Figura 29 se ilustra de forma muy general el proceso de inicialización.

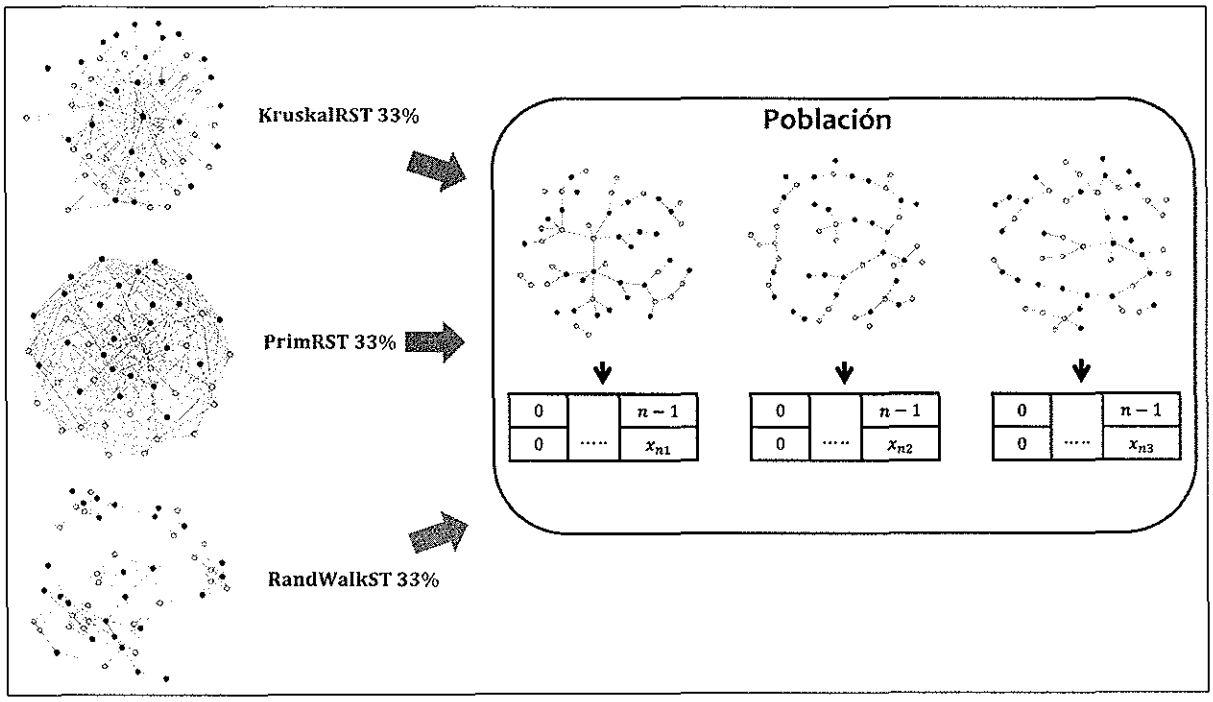


Figura 29: Población inicial en Dandelion.

V.5.2. Operadores evolutivos

Los operadores evolutivos que utiliza dandelion se definen con base en los estudios experimentales hechos por Paulden y Smith (2006b). En el caso de los operadores de recombinación, Paulden y Smith (2006b) comparan las recombinaciones de un punto de cruce (*one-point crossover*), dos puntos de cruce (*two-point crossover*) y cruce uniforme (*uniform crossover*). Ellos demuestran en forma empírica que, en promedio, la recombinación de un punto de cruce tiene mejor herencia persistente. Para los operadores de mutación, comparan la mutación de un solo elemento (*single-element mutation*) y la mutación por intercambio (*exchange mutation*) y demuestran que dandelion tiene una localidad acotada bajo estos operadores; sin embargo, concluyen que la mutación de un solo elemento se debe usar en preferencia de la mutación por intercambio, ya que es significativamente menos perturbadora, es decir tiene mejor localidad. Este análisis lo realizan obteniendo la distancia de Hamming entre el árbol original y el mutado. Con base en lo anterior se decide implementar la recombinación de un punto de cruce y la mutación de un solo elemento.

Recombinación de un punto de cruce

La recombinación de un punto (*one-point crossover*) es de los operadores de recombinación que más se conocen; este operador se describe en Eiben y Smith (2003); además, este operador es el que utiliza Holland (1992) en su algoritmo genético canónico. Este operador trabaja seleccionando primero una posición aleatoria k del genotipo, para después dividir a ambos padres en este punto creando dos hijos al intercambiar sus posiciones a partir del punto k. Es decir que para el hijo 1, se copian las primeras k posiciones del padre 1 y las restantes del padre 2. Para el hijo 2, se realiza de forma

similar pero intercambiando los padres; en la Figura 30 se muestra un ejemplo de esta recombinación con dandelion. Cabe mencionar que la mutación se efectúa en el genotipo dandelion y luego se realiza la cirugia de árbol para generar el árbol mutado.

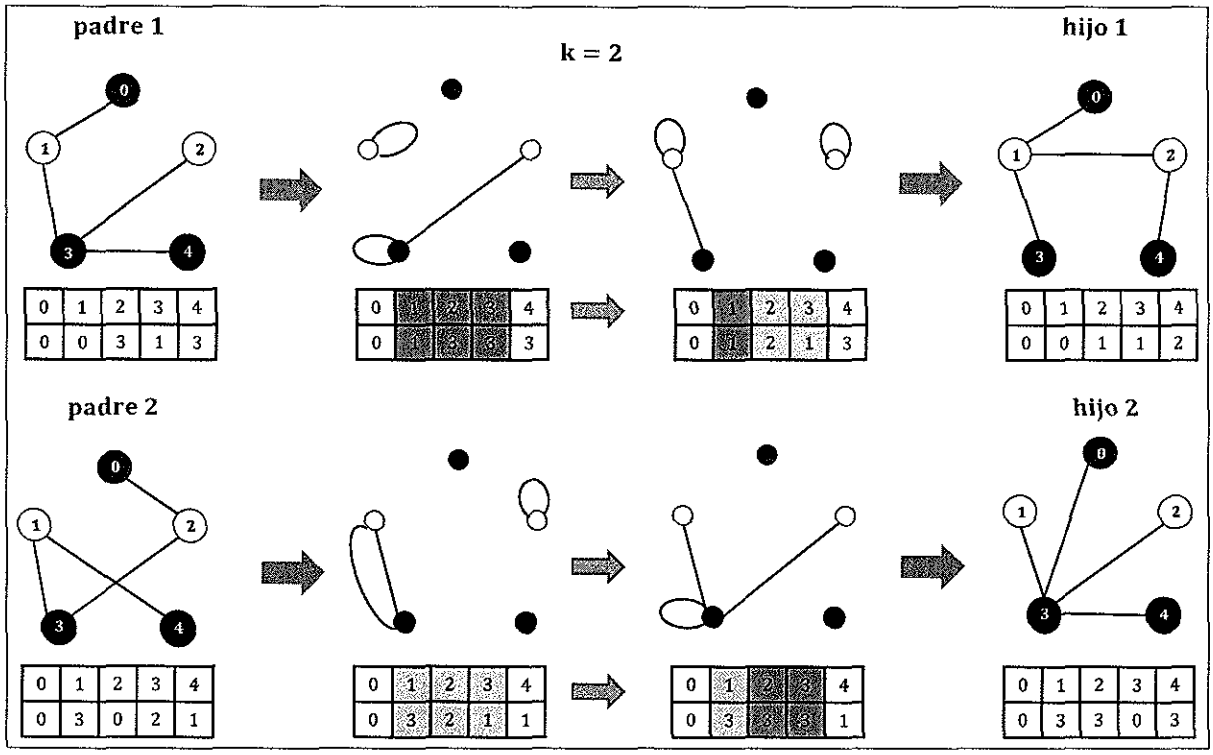


Figura 30: Recombinación de un punto para Dandelion.

Mutación de un solo elemento

La mutación de un solo elemento (*single element mutation*) para dandelion primero selecciona un índice aleatorio de dandelion (randomIndex $\in [1, n-2]$) y un identificador aleatorio de un vértice (randomVertexId $\in [0, n-1]$); una vez que se definen estos dos valores, simplemente se escribe en el índice aleatorio el identificador del vértice. En la Figura 31 se observa un ejemplo de este operador con randomIndex $= 2$ y randomVertexId $= 4$.

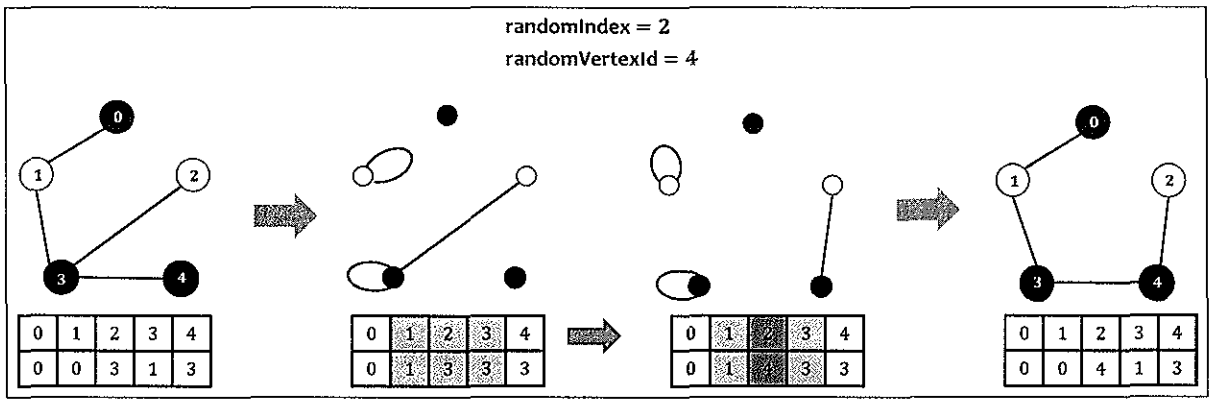


Figura 31: Mutación de un solo elemento para Dandelion.

V.6. Algoritmos híbridos

Durante el desarrollo de esta investigación, el autor trabajó en equipo con Daniel Fajardo y Daniel Brubeck para crear un simulador para el PAG. Cada miembro del equipo desarrolló una metaheurística con distintos enfoques para generar soluciones al PAG. Aprovechando este simulador, se implementó un híbrido que combina el algoritmo determinístico (Fajardo y Fernández, 2010) existente con el algoritmo genético, con el objetivo de analizar si el algoritmo híbrido genera mejores soluciones que el algoritmo genético. En esta sección se describe brevemente los tipos de híbridos que se implementan.

V.6.1. Tipos de híbridos propuestos

Se proponen tres tipos de algoritmos híbridos. Los primeros dos algoritmos son en cascada, básicamente se ejecuta un algoritmo después de otro, mientras que el tercero es intercalado con un criterio de terminación con base en tiempo o valor de la aptitud del mejor individuo de la población (función objetivo). Suponiendo que el tamaño de la población del AG es X_n, el primer tipo de algoritmo híbrido ejecuta el algoritmo deter-

minístico X_n veces y después se ejecuta el algoritmo genético tomando como población inicial los X_n resultados del algoritmo determinístico; en la Figura 32 se observa un esquema general del algoritmo híbrido 1.

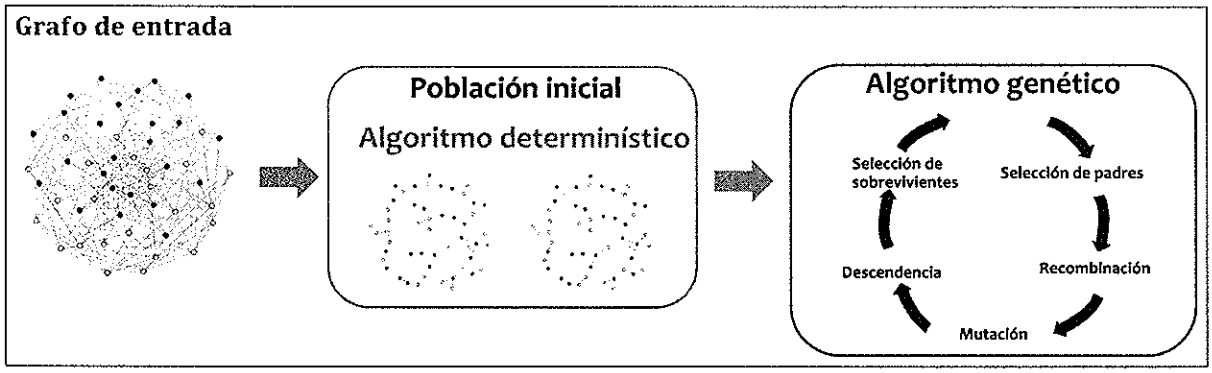


Figura 32: Esquema general del algoritmo híbrido 1.

Para el segundo tipo de algoritmo, primero se ejecuta el algoritmo genético y la salida de este algoritmo se vuelve la entrada del algoritmo determinístico; la Figura 33 ilustra este proceso.

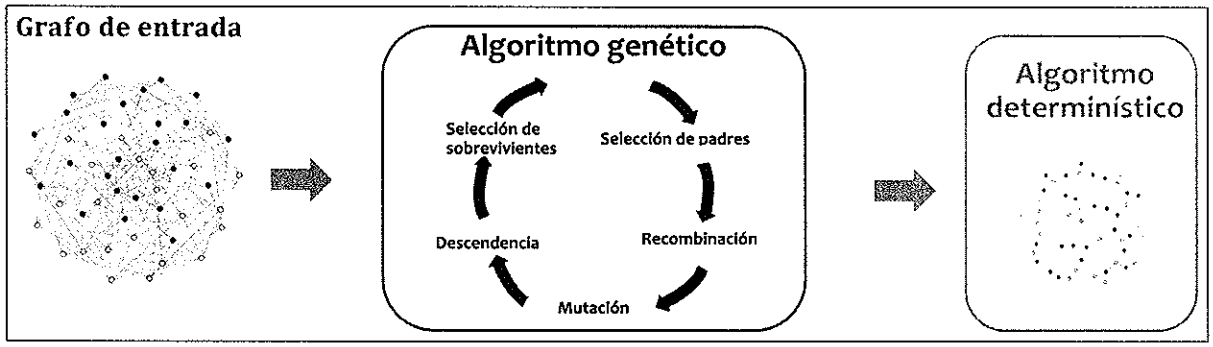


Figura 33: Esquema general del algoritmo híbrido 2.

Por último, el algoritmo híbrido 3 o intercalado, es cuando se ejecuta intercaladamente el algoritmo determinístico y el algoritmo genético hasta alcanzar un criterio de finalización por tiempo o aptitud global (función objetivo). En la Figura 34 se observa el esquema general de este algoritmo.

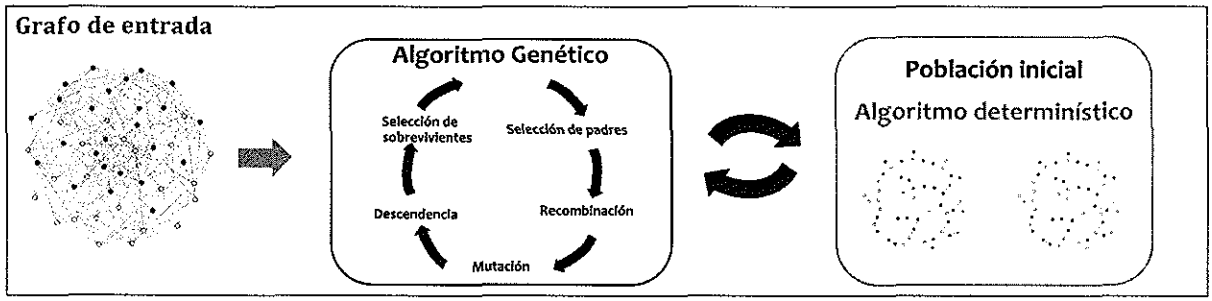


Figura 34: Esquema general de híbrido intercalado.

Estos tres algoritmos híbridos se implementan con distintas variantes, que consisten en distintos tamaños de la población y distintas probabilidades de ocurrencia para sus operadores de variación. Estas variantes se comparan en el Capítulo VI para determinar la mejor configuración del algoritmo híbrido. Si se utiliza el AG simple para el algoritmo híbrido intercalado, el tiempo de ejecución será demasiado grande ya que se ejecuta varias veces el algoritmo genético; para resolver esta problemática, se propone utilizar un micro algoritmo genético, el que se puede observar en la Sección V.6.2.

V.6.2. Micro algoritmo genético

Krishnakumar (1989) diseñó un AG con una población de 5 individuos; a su algoritmo de representación binaria lo nombró micro algoritmo genético (MGA). Krishnakumar utilizó elitismo para preservar la mejor cadena que se encuentra al término de una convergencia, como uno de los individuos obligatorios para la siguiente generación. Al comparar el desempeño del MGA contra un AG simple con una población de 50 individuos, Krishnakumar obtuvo mejores resultados para funciones de un solo objetivo; además, el comprueba que el AG de población reducida convergía más rápido. Coello y Pulido (2001) diseñaron un MGA para resolver problemas de optimización de múltiples objetivos. Se trabaja con una población de cuatro individuos y utiliza una

memoria secundaria que almacena las soluciones potenciales (no dominadas) a lo largo de la búsqueda. El MGA que se implementa en esta tesis utiliza los mismos operadores de variación descritos anteriormente, pero solamente para la representación EdgeSet. La única variante con el algoritmo genético simple es el tamaño de la población y que utiliza métodos de perturbación que se describen en la Sección V.6.3

V.6.3. Métodos de perturbación

Una desventaja de los MGAs, es que si un individuo de la población sobresale con respecto al resto de la población, la población tiende a converger a este individuo, generando como resultado que todos los individuos sean exactamente iguales. A esto se le llama 'convergencia prematura' o también se puede decir que el algoritmo se queda atrapado en un óptimo local ya que los operadores de mutación no son lo suficientemente perturbadores para escapar del óptimo local. Las principales causas de que ocurra una convergencia prematura en un algoritmo genético es por una presión de selección muy alta, probabilidad de mutación muy baja o un tamaño de población muy pequeño, como es el caso del MGA. Si se utiliza una medida de similaridad entre los individuos dada por la distancia de Hamming, se pueden plantear algunas formas de evitar la convergencia prematura, previniendo el incesto (evitar la recombinación de individuos similares), haciendo un reemplazo prioritario de individuos similares o aumentando el tamaño de la población. Para tratar de salir de estos óptimos locales en el MGA se proponen distintos métodos de perturbación de la población. Estos métodos perturban toda o la mayoría de la población al aplicar alguna heurística. Los métodos de perturbación que se aplican se describen a continuación:

Perturbación forzada

La idea del método de pertubación forzada (PERT) es adelantar en varias generaciones el ciclo evolutivo aplicando los operadores de variación de forma obligada para cada individuo en la población. Este método funciona primero al clonar la población (que es el mismo individuo), para después aplicar mutación a todos los individuos de la población clonada. En este punto se tienen dos poblaciones, la original y la clonada, a partir de éstas, se genera una tercera población por medio del operador de recombinación tomando un padre de la población original y uno de la población clonada. Por último, se agregan todos los individuos mutados y los descendientes a la población original, para continuar con el ciclo evolutivo. Nótese que para este método, el tamaño de la población crece a más del doble, pero el criterio de selección de sobrevivientes la reduce a su tamaño normal al eligir los mejores.

Reinicio aleatorio de la población

El método de reinicio aleatorio de la población (RAND) realiza una reiniciación parcial de la población con los métodos para generar árboles aleatorios vistos en la Sección V.1.1. Es decir, que el 90 % de los individuos de la población se reemplazan por individuos generados aleatoriamente. Esto se hace para no perder este individuo sobresaliente y además generar diversidad en la población.

Peturbación forzada con reinicio aleatorio

El método de perturbación forzada con reinicio aleatorio (PERT-RAND) combina los dos anteriores, es decir, a la mitad de la población se le aplica la perturbación forzada; mientras que a la otra mitad se le reinicia de forma aleatoria.

Aplicar el algoritmo determístico

En este método de perturbación lo que se hace es aplicar el algoritmo determinístico existente a cada individuo de la población. El algoritmo determinístico *CBAP*, dependiendo su configuración, puede generar distintas salidas para una misma entrada. Entonces lo que se busca es mejorar este individuo. Un problema de este método es que si el individuo es un equilibrio, el *CBAP* no genera una mejora en el individuo (Fajardo y Fernández, 2010). A este método se le llama *PCBAP*, y es basicamente el que se utiliza en forma conjunta con el algoritmo genético para el híbrido intercalado.

Capítulo VI

Experimentos preliminares

En este capítulo se describen los experimentos preliminares que se realizaron para obtener la mejor configuración de los operadores evolutivos del algoritmo genético, así como las mejores variantes del algoritmo híbrido. Las representaciones con las cuales se experimenta son NetKeys y EdgeSet, para el caso de Dandelion se hace un análisis de las soluciones generadas. Además se determina cuál es el enfoque que obtiene mejores resultados de los algoritmos determinísticos conocidos. Se utilizan los grafos descritos en el Capítulo IV como entradas de los experimentos preliminares; cada grafo tiene 51 vértices y un número de aristas entre 130 y 190 (son grafos dispersos). En algunos casos se utilizan grafos de 65 vértices y de distintas densidades. En la Tabla X se muestra la configuración general del algoritmo genético que se utiliza a lo largo de este capítulo; los únicos parámetros que varían para algunas comparaciones son la representación y los operadores de variación (los demás parámetros no cambian). La probabilidad de recombinación y de mutación es de 0.85 y 0.3, respectivamente; éstos son valores que se utilizan comúnmente en algoritmos genéticos. Se utiliza un tamaño de población de 200 individuos y un criterio de finalización de 1000 generaciones; lo anterior para asegurar que el algoritmo explore el espacio de soluciones de la manera más completa posible. Finalmente, los métodos de selección de padres y sobrevivientes que se utilizan se explican en la Sección V.2.

VI.1. Comparación de los operadores de variación para la representación NetKeys

En esta sección se realiza una comparación empírica entre los operadores de variación de NetKeys que se describen en la Sección V.3.2. La configuración inicial para el algoritmo genético se muestra en la Tabla XI.

Las Tablas XII-XIV presentan los resultados de 30 ejecuciones por grafo, utilizando el promedio y el valor más grande de la función objetivo de la población, además del máximo tiempo en milisegundos que tarda una ejecución. La única diferencia entre estas tablas es que se varía el número de veces en que se aplica la mutación por cambio de peso, utilizando 5, 10 y 20 veces, respectivamente; en todas las tablas se aplican los tipos de recombinación aritmética sencilla, simple y completa; tanto el operador de mutación como los operadores de recombinación para NetKeys se describen en la Sección V.3.2. Las celdas con números en negritas en las Tablas XII-XIV representan los valores más altos de los promedios y valores máximos de la función objetivo por cada tipo de grafo. El fondo gris en algunas celdas de las tablas, indican los valores más altos de los promedios y valores máximos de la función objetivo por grafo de cualquiera de las tres tablas. Se observa que al aplicar la mutación de cambio de peso 5 veces con la recombinación aritmética simple se obtienen los mejores resultados. Esta última observación parece lógica, si se considera que la recombinación aritmética completa genera dos descendientes iguales, algo que no es muy deseable en un algoritmo genético, ya que esto hace que se reduzca la diversidad de la población. La recombinación aritmética sencilla modifica solo un alelo del genotipo de los padres en los hijos, por lo que los genotipos de los padres y descendientes son distintos; sin embargo, pueden ser iguales en el árbol de esparcimiento que representan.

Tabla X: Configuración general del algoritmo genético.

Probabilidad de recombinación	0.85
Probabilidad de mutación	0.3
Selección de padres	Probabilidad de recombinación aleatoria
Selección de sobrevivientes	Basado en aptitud $(\mu + \lambda)$
Tamaño de la población	200
Número de generaciones	1000
Criterio de finalización	Por generaciones

Tabla XI: Configuración del algoritmo genético con la representación NetKeys.

Representación	Network random keys
Operador de recombinación	Aritmética sencilla, simple y completa
Operador de mutación	Cambio de peso aleatorio (5, 10 y 20)

Tabla XII: Comparación de las recombinaciones aritméticas sencilla, simple y completa, aplicando 5 veces la mutación NetKeys de cambio de peso. Se resalta en negrita los valores más altos de los promedios y valores máximos de la función objetivo por cada tipo de grafo. El fondo gris de las celdas indica los valores más altos de los promedios y valores máximos de la función objetivo por grafo de cualquiera de las Tablas Tablas XII-XIV.

Tipos de grafos	Recomb. Aritmética Sencilla			Recomb. Aritmética Simple			Recomb. Aritmética Completa		
	Función Objetivo		Tiempo (ms)	Función Objetivo		Tiempo (ms)	Función Objetivo		Tiempo (ms)
	Prom.	Max.	Max.	Prom.	Max.	Max.	Prom.	Max.	Max.
barabasi 1	1502.3	1546	55603	**1516.6**	**1599**	55586	1508.1	1555	60396
barabasi 2	1560.1	1604	58369	**1579.9**	**1628**	58687	1572.4	1614	62896
erdoreny 1	1708.7	1773	61990	**1711.5**	**1791**	62525	1709.6	1781	68078
erdoreny 2	1730.9	1809	64762	1732.4	**1810**	64298	**1734.0**	1788	69937
evolstoc	1679.5	1743	66248	**1689.0**	**1750**	66597	1678.5	1741	72592
geometri	1529.5	1588	60646	**1537.9**	**1622**	61044	1520.3	1580	65976
gradoady	1729.0	1779	68340	**1738.8**	1826	68363	1732.4	**1838**	75046
watzstro	1693.6	**1776**	56889	**1701.9**	1754	56938	1691.4	1766	61905
erdoreny 1*	1802.6	1858	63751	**1817.6**	**1872**	63921	1807.5	1861	69467

Tabla XIII: Comparación de las recombinaciones aritméticas sencilla, simple y completa, aplicando 10 veces la mutación NetKeys de cambio de peso. Se resalta en negrita los valores más altos de los promedios y valores máximos de la función objetivo por cada tipo de grafo. El fondo gris de las celdas indica los valores más altos de los promedios y valores máximos de la función objetivo por grafo de cualquiera de las Tablas Tablas XII-XIV.

Tipos de grafos	Recomb. Aritmética Sencilla			Recomb. Aritmética Simple			Recomb. Aritmética Completa		
	Función Objetivo		Tiempo (ms)	Función Objetivo		Tiempo (ms)	Función Objetivo		Tiempo (ms)
	Prom.	Max.	Max.	Prom.	Max.	Max.	Prom.	Max.	Max.
barabasi 1	1505.4	**1570**	55968	**1514.9**	1550	56118	1509.9	1559	60837
barabasi 2	1550.8	1597	58648	1561.3	1598	58795	**1566.7**	**1629**	63957
erdoreny 1	1687.2	1755	62398	**1693.2**	**1778**	63685	1687.1	1737	68572
erdoreny 2	1721.7	1782	64940	**1733.8**	1788	65274	1721.2	**1795**	70592
evolstoc	**1673.7**	1734	66809	1671.9	1722	67174	1662.9	**1736**	73009
geometri	1533.8	**1628**	61132	**1537.6**	1606	60945	1530.3	1597	67663
gradoady	**1732.2**	1792	68735	1705.5	1776	69108	1710.0	**1804**	75752
watzstro	1677.0	1727	57220	**1695.3**	**1741**	57408	1682.0	1731	62661
erdoreny 1*	**1799.9**	**1860**	66107	1789.5	1843	64209	1795.7	1854	70339

Tabla XIV: Comparación de las recombinaciones aritméticas sencilla, simple y completa, aplicando 20 veces la mutación NetKeys de cambio de peso. Se resalta en negrita los valores más altos de los promedios y valores máximos de la función objetivo por cada tipo de grafo. El fondo gris de las celdas indica los valores más altos de los promedios y valores máximos de la función objetivo por grafo de cualquiera de las Tablas Tablas XII-XIV.

Tipos de grafos	Recomb. Aritmética Sencilla			Recomb. Aritmética Simple			Recomb. Aritmética Completa		
	Función Objetivo		Tiempo (ms)	Función Objetivo		Tiempo (ms)	Función Objetivo		Tiempo (ms)
	Prom.	Max.	Max.	Prom.	Max.	Max.	Prom.	Max.	Max.
barabasi 1	**1498.8**	**1536**	56544	1495.4	1523	56845	1476.7	1519	61836
barabasi 2	1528.8	1572	59451	**1538.2**	1577	59453	1522.8	**1590**	64573
erdoreny 1	1649.5	1727	63126	**1649.0**	**1708**	63587	1618.7	1672	69498
erdoreny 2	**1667.5**	**1790**	64909	1665.6	1724	65632	1641.9	1726	71508
evolstoc	**1629.9**	1669	66925	1617.1	**1692**	67980	1599.5	1666	73722
geometri	1524.0	1595	61526	**1526.9**	**1623**	61649	1501.4	1562	67692
gradoady	1660.1	1741	69225	**1662.8**	**1749**	69682	1633.8	1729	76424
watzstro	1639.4	**1716**	57767	**1645.0**	1707	58010	1627.7	1685	62779
erdoreny 1*	**1741.1**	1789	64806	1737.5	**1797**	64814	1717.8	1792	70853

En el caso de la mutación si se aplica solo 1 vez sucede un efecto similar que al aplicar la recombinación aritmética simple, es decir, que el árbol que representa el genotipo original y el mutado sean el mismo, es por ello que se experimentó con 5, 10 y 20, siendo 5 el número de aplicaciones con las que se obtienen mejores resultados; al parecer, al aplicar más veces el número de mutaciones se perturba demasiado el genotipo y se pierden las buenas características del individuo.

VI.2. Comparación de los operadores de variación para la representación EdgeSet

En esta sección se realiza una comparación empírica entre los operadores de mutación de EdgeSet que se describen en la Sección V.4.2. En la Tabla XV se muestra la configuración inicial del algoritmo genético que se utiliza en esta sección. Los resultados de la comparación se presentan en la Tabla XVI, para generar la tabla se utilizan los resultados de 30 ejecuciones por grafo, los datos que se muestran en la tabla son el promedio y el valor más alto de la máxima función objetivo en las 30 ejecuciones, además de la desviación estandar del máximo tiempo y el máximo tiempo en milisegundos.

A continuación se calcula una cota superior para el óptimo global para un grafo de 51 vértices, es decir el máximo valor que se puede obtener en estos experimentos de acuerdo a la Ecuación 6 que se menciona en la Sección III.3:

$$\frac{(n-1)(3n-1)}{4} = \frac{(51-1)(3(51)-1)}{4} = \frac{(50)(152)}{4} = 1900$$

En la Tabla XVI se resalta en negrita los valores más altos de los promedios y valores máximos de la función objetivo por cada tipo de grafo, por lo que se observa que la mutación que obtiene mejores resultados es la de borrado aleatorio de arista, incluso

Tabla XV: Configuración de algoritmo genético con la representación EdgeSet.

Representación	EdgeSet
Operador de recombinación	Unión de árboles
Operador de mutación	Mutación por borrado y adición aleatoria de arista

Tabla XVI: Comparación de las mutaciónes de borrado y adición aleatoria de arista para EdgeSet. Se resalta en negrita los valores más altos de los promedios y valores máximos de la función objetivo por cada tipo de grafo.

Tipos de grafos	Borrado aleatorio de arista				Adición aleatoria de arista			
	Función Objetivo		Tiempo (ms)		Función Objetivo		Tiempo (ms)	
	Prom.	Max.	Desv.	Max.	Prom.	Max.	Desv.	Max.
barabasi 1	**1537.9**	**1577**	13.7	41209	1520.7	1553	22.6	36657
barabasi 2	**1608.9**	**1641**	19.5	41205	1511.2	1537	10.3	37992
barabasi 1*	**1838.1**	**1882**	27.8	43472	1660.1	1721	19.2	40808
barabasi 2*	1663.8	**1892**	55.1	41739	**1676.6**	1731	20.3	44112
completo	**1749.3**	**1900**	81.2	81875	1679.6	1757	29.8	62133
evolstoc	**1773.6**	**1806**	23.3	39404	1592.2	1652	14.6	44459
evolstoc *	**1890.2**	**1900**	5.2	41176	1770.7	1820	17.3	41389
geometri	**1803.3**	**1896**	49.0	42476	1704.3	1776	21.8	41811
geometri *	**1625.5**	**1690**	43.1	43266	1579.0	1625	20.8	49155

para el caso del grafo de evolución estocástica y el grafo completo se obtiene el óptimo global de 1900.

VI.3. Análisis de validez de los árboles solución que se generan con la representación Dandelion

En esta sección se analizan los operadores evolutivos de Dandelion que se describen en la Sección V.5.2. El análisis se realiza porque al examinar los árboles que genera el algoritmo genético con representación Dandelion se obtienen algunas soluciones inválidas. Validez se refiere a que dado un grafo de entrada $G = (V, E)$ y un árbol solución generado $T = (V, E')$, el árbol T es válido si el conjunto de aristas del árbol es un subconjunto de las aristas del grafo de entrada $E' \subseteq E$.

Los métodos que pueden generar soluciones no válidas en Dandelion son los operadores de variación, por lo que en esta sección se realiza un análisis empírico para ver la validez de los árboles que se obtienen cuando se aplica mutación y recombinación. Los datos del análisis se obtienen al hacer lo siguiente: se utilizan los grafos aleatorios del Capítulo IV, se generan 1000 árboles por cada tipo de grafo, los árboles generados se mapean a la representación Dandelion, se les aplican los operadores de variación de forma independiente, por último, se evalúa la validez de cada árbol que se obtiene al comparar las aristas del árbol con las aristas del grafo de entrada. Este procedimiento se utiliza para la Tabla XVII. Cabe destacar que para esta tabla solo se utilizan grafos dispersos con 51 vértices. En la Tabla XVII se puede observar que el porcentaje de árboles válidos cuando se aplica el operador de recombinación es muy bajo, excepto para los grafos Barabási-Albert; en el caso de la mutación siempre se obtienen árboles

válidos.

Para las Tablas XVIII-XX se utilizan grafos de distintas densidades con 65 vértices de tipo Barabási-Albert, Erdös Rényi y de evolución estocástica, respectivamente; en estas tablas se utiliza el mismo procedimiento que para la Tabla XVII pero para 2000 árboles. En todas las tablas se puede observar que el operador de mutación de Dandelion no genera árboles inválidos; por el contrario, el operador de recombinación tiende a generar soluciones inválidas, aunque conforme aumenta la densidad del grafo de entrada se reduce la probabilidad de generar un árbol inválido. Con base en este análisis, la representación Dandelion sólo se puede comparar con EdgeSet y NetKeys para el caso del grafo completo, ya que para grafos no completos se tendría que aplicar un mecanismo de reparación después del operador de recombinación o aplicarlo muchas veces hasta obtener una solución válida.

VI.4. Comparación del algoritmo determinístico con enfoque cooperativo y no cooperativo

Fajardo y Fernández (2010) proponen un algoritmo determinístico para obtener soluciones aproximadas al PAG, con base en dos enfoques (cooperativo y no cooperativo) y al que nombran CBAP. El enfoque cooperativo es aquel en donde todos los vértices tratan de mejorar la ganancia global de la solución, y el enfoque no cooperativo es aquel en donde cada vértice prefiere mejorar su ganancia individual por encima de la ganancia global. En esta sección se compara el algoritmo CBAP con estos dos enfoques para definir cuál es el enfoque que obtiene mejores resultados. El algoritmo CBAP trabaja a partir de un árbol inicial e intenta mejorarlo intercambiando aristas,

Tabla XVII: Análisis del número y porcentaje de árboles válidos generados con los operadores de mutación y recombinación de Dandelion utilizando grafos dispersos.

Tipos de grafos	Num. Árboles válidos		Prob. Árboles válidos	
	Mutación	Recombinación	Mutación	Recombinación
barabasi 2*	1000	422	1	0.422
barabasi 2	1000	385	1	0.385
completo	1000	1000	1	1
erdoreny 1*	1000	118	1	0.118
erdoreny 1	1000	78	1	0.078
erdoreny 2*	1000	93	1	0.093
erdoreny 2	1000	115	1	0.115
evolstoc *	1000	90	1	0.090
evolstoc	1000	83	1	0.083
geometri *	1000	178	1	0.178
geometri	1000	285	1	0.285
gradoady *	1000	57	1	0.057
gradoady	1000	64	1	0.064
watzstro	1000	42	1	0.042
watzstro *	1000	70	1	0.070

Tabla XVIII: Análisis del número y porcentaje de árboles válidos generados con los operadores de mutación y recombinación de Dandelion utilizando grafos Barabási-Albert con distintas densidades.

Densidad	Num. Árboles válidos		Prob. Árboles válidos	
	Mutación	Recombinación	Mutación	Recombinación
128	2000	815	1	0.4075
256	2000	629	1	0.3145
512	2000	594	1	0.297
1024	2000	812	1	0.406
2048	2000	1891	1	0.9455

Tabla XIX: Análisis del número y porcentaje de árboles válidos generados con los operadores de mutación y recombinación de Dandelion utilizando grafos Erdös Rényi con distintas densidades.

Densidad	Num. Árboles válidos		Prob. Árboles válidos	
	Mutación	Recombinación	Mutación	Recombinación
128	2000	85	1	0.0425
256	2000	102	1	0.051
512	2000	363	1	0.1815
1024	2000	439	1	0.2195
2048	2000	1895	1	0.9475

Tabla XX: Análisis del número y porcentaje de árboles válidos generados con los operadores de mutación y recombinación de Dandelion utilizando grafos evolutivos estocásticos con distintas densidades.

Densidad	Num. Árboles válidos		Prob. Árboles válidos	
	Mutación	Recombinación	Mutación	Recombinación
128	2000	203	1	0.1015
256	2000	143	1	0.0715
512	2000	222	1	0.111
1024	2000	421	1	0.2105
2048	2000	1847	1	0.9235

hasta llegar a un equilibrio. En la Tabla XXI se muestra la comparación del enfoque cooperativo y el no cooperativo. Para el árbol inicial de ambos enfoques se utilizan los algoritmos para generar árboles de esparcimiento aleatorios *KruskalRST*, *PrimRST* y *RandWalk* (ver Sección V.1.1). En la Tabla XXI se muestran el promedio y el valor más alto de la máxima función objetivo de 30 ejecuciones por grafo, además se resaltan con negrita los valores más altos por cada grafo. Con base en esta tabla se observa que el enfoque cooperativo genera mejores resultados y, que al utilizar *PrimRST* como árbol inicial, tiene un mejor desempeño que con otros árboles iniciales.

VI.5. Análisis y comparaciones de las variantes del algoritmo híbrido

En esta sección se hace una comparación empírica de los tipos de híbridos propuestos (ver Sección V.6). En este trabajo se proponen tres tipos de híbridos, dos de ellos son en "cascada", es decir se ejecuta un algoritmo y después otro, mientras que el tercero es intercalado con un criterio de terminación con base al tiempo de ejecución o valor de la aptitud del mejor individuo de la población.

Tabla XXI: Comparación del algoritmo determinístico entre el enfoque cooperativo y no cooperativo.

Tipos de grafos	CBAP Cooperativo						CBAP No Cooperativo					
	KruskalRST		PrimRST		RandWalk		KruskalRST		PrimRST		RandWalk	
	Prom.	Max.	Prom.	Max.	Prom.	Max.	Prom.	Max.	Prom.	Max.	Prom.	Max.
barabasi 2*	**1690.7**	1840	1689.5	1825	1677.3	**1847**	1506.3	1613	1502.2	1612	1509.1	1623
barabasi 1*	1683.2	1806	**1689.1**	1798	1675.7	**1808**	1498.6	1604	1506.6	1604	1505.8	1604
barabasi 1	1427.7	1511	1426.5	1495	1429.1	1511	1427.6	1505	1429.9	1496	**1431.3**	**1516**
barabasi 2	1494.5	**1585**	1492.7	1562	1493.7	**1585**	1499	1574	1501	1575	**1501.7**	1562
completo	1682.9	1828	**1730.1**	**1880**	1670.2	1822	1600	1600	1600	1600	1600	1600
erdoreny 1*	1706.8	1843	**1725.9**	**1860**	1701.7	1846	1591.1	1636	1589.3	1637	1590.7	1635
erdoreny 1	1604.6	1709	**1615.9**	**1721**	1601.3	1717	1537.8	1592	1534.1	1599	1536.8	1602
erdoreny 2*	1709.5	**1851**	**1729.5**	1844	1715.7	1843	1586.4	1620	1586.5	1620	1588.4	1620
erdoreny 2	1633	**1806**	**1648.5**	1780	1625.4	1764	1547.3	1614	1548.6	1619	1547.9	1619
evolstoc *	1773.2	1875	**1790**	**1886**	1766	1878	1585.9	1630	1583.5	1628	1585.3	1627
evolstoc	1582.1	1700	1583.7	**1712**	1585.4	1681	1619.8	1697	1619.2	1697	**1621.2**	1680
geometri *	1540.3	1743	1560.8	1732	1534.5	1733	1672.9	1779	1667	**1780**	**1675.1**	1778
geometri	**1436**	**1621**	1431.6	1588	1433.5	1614	1406.7	1478	1402.7	1482	1400.3	1480
gradoady *	1697.3	1799	**1708.2**	**1835**	1691.5	1813	1574.4	1619	1573.8	1616	1572.9	1616
gradoady	1636.1	1775	**1649.7**	**1779**	1628	1763	1577.5	1649	1576.1	1642	1574.6	1639
watzstro	**1574.7**	1731	1570	**1741**	1564.8	1713	1559.6	1660	1559.7	1653	1562.2	1654
watzstro *	1709.1	**1853**	**1726**	1840	1698.5	1837	1581.7	1635	1583.9	1632	1580.9	1633

VI.5.1. Comparación de los tipos de algoritmos híbridos en cascada

En esta sección se analizan algunas variantes que se plantean para los dos tipos de híbridos en cascada. Para cada uno de estos híbridos se proponen tres variantes. En el primer tipo de híbrido o *HGA*-1 (*Hybrid Genetic Algorithm*) primero se ejecuta un número determinado de veces el algoritmo determinístico, y después se ejecuta el algoritmo genético tomando como población inicial los árboles resultantes de las ejecuciones del algoritmo determinístico. Las variantes para este tipo de híbrido se basan en definir cómo se genera la población inicial del algoritmo genético. Sea X_n el tamaño de la población del algoritmo genético. Para la primer variante, que se denomina *HGA*-1_1, se ejecuta el algoritmo *CBAP* X_n veces; para la segunda variante, *HGA*-1_2, se ejecuta el *CBAP* $4X_n$ veces, guardando en los X_n mejores individuos; para la última variante, *HGA*-1_3, se genera la mitad de la población inicial con el *CBAP* y la otra mitad con métodos para generar árboles aleatorios. En la segunda parte de las tres variantes del *HGA*-1 se ejecuta el algoritmo genético con la configuración que se muestra en la Tabla X.

En la Figura 35 se presenta una gráfica comparativa de los máximos valores de la función objetivo promedio y el tiempo promedio de cada uno de los 17 tipos de grafos con los que se experimenta, los cuales son los 9 que se observan en la Tabla V más sus variantes con el óptimo global insertado. Con esta gráfica no es muy claro determinar qué variante es la mejor, por lo que se realiza un análisis estadístico. Este análisis estadístico, que se realiza por pares de muestras, sirve para determinar si existe una diferencia significativa entre dos muestras de datos y en caso de que exista una diferencia, determina cuál es mayor que otra (ver Apéndice A).

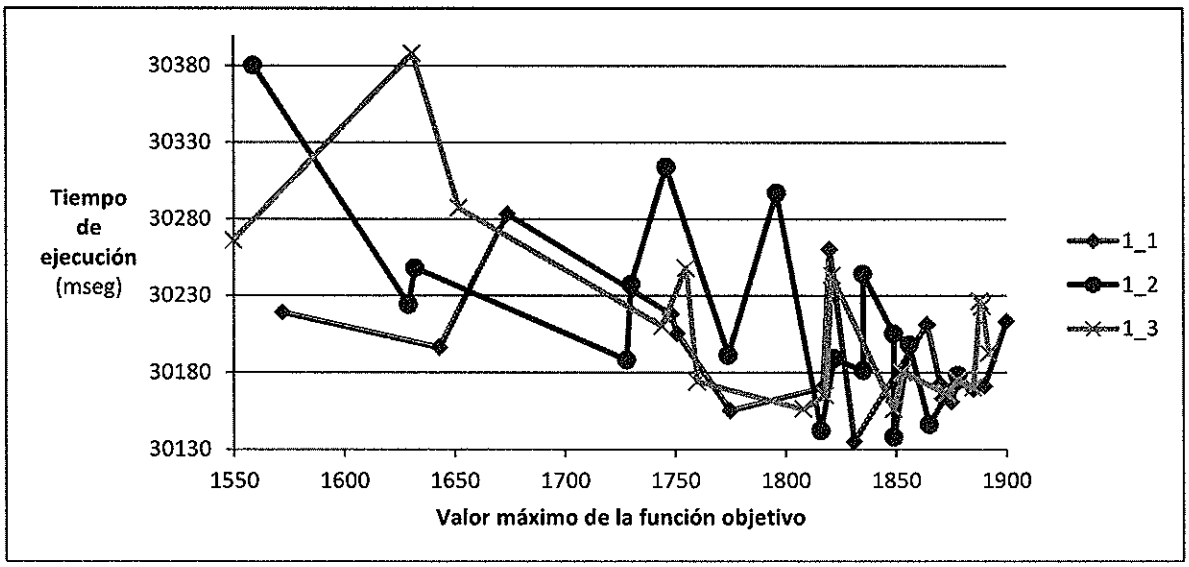


Figura 35: Valor máximo de la función objetivo vs tiempo de ejecución de las variantes del algoritmo híbrido 1.

Para el híbrido 2 (o HGA-2) primero se ejecuta el algoritmo genético y después el algoritmo determinístico *CBAP*. Son tres las variantes para este tipo de híbrido, las cuales son simplemente cambios en el número máximo de generaciones o en el tamaño de la población a generar para el algoritmo genético. La primer variante, *HGA*-2_1, ejecuta el algoritmo genético 1000 generaciones como máximo con una población de 200 individuos; ésta es la configuración que por lo general se utiliza. La segunda variante, *HGA*-2_2, es con 1000 generaciones y una población de 100 individuos. La tercer variante, *HGA*-2_3, es con 500 generaciones y 100 individuos. En la Figura 36 se muestra una gráfica comparativa de los máximos valores de la función objetivo promedio y el tiempo promedio de cada uno de los 17 tipos de grafos con los que se experimenta. Se observa que la variante *HGA*-2_2 es la que más tiempo en promedio necesita, en cuanto al valor más alto de la función objetivo, no se observa una mejor que otra con claridad. Comparando las figuras 35 y 36 se puede ver que el *HGA*-1 en sus tres variantes es más rapido (cada ejecución tarda alrededor de 30000 milisegundos) que el *HGA*-2

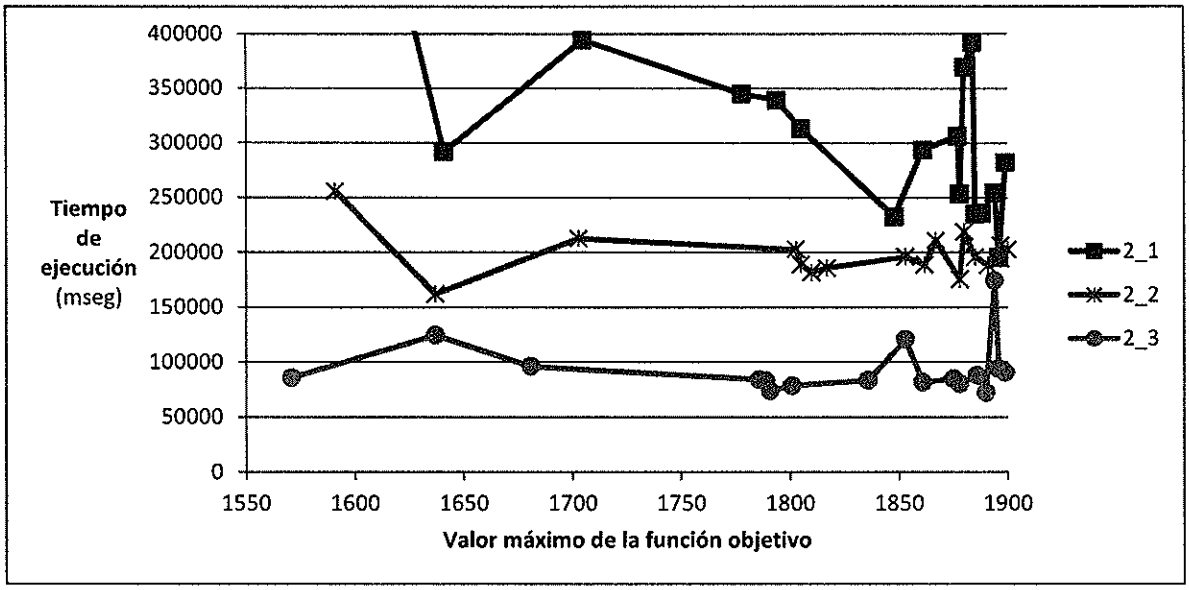


Figura 36: Valor máximo de la función objetivo vs tiempo de ejecución de las variantes del algoritmo híbrido 2.

Tabla XXII: Posición al evaluar con multiples análisis estadísticos la mayor ganancia al evaluar la función objetivo y el menor tiempo de las variantes del híbrido en cascada.

Variante de híbrido	Mayor ganancia	Menor tiempo
HGA-1_1	1	1
HGA-1_2	6	2
HGA-1_3	4	1
HGA-2_1	3	5
HGA-2_2	2	4
HGA-2_3	5	3

(sus ejecuciones son mayores a 60000 milisegundos). Para confirmar esto y definir cuál es el mejor híbrido en cascada se realiza un análisis estadístico de todas las variantes de los híbridos en cascada. Las posiciones al evaluar el análisis estadístico se pueden observar en la Tabla XXII. Lo más destacable en esta tabla es que el mejor híbrido tanto en la máxima ganacia de las soluciones generadas y en el menor tiempo es la variante *HGA*-1. Las variantes del *HGA*-2 son las que mayor tiempo necesitan y sus soluciones son peores al evaluar su función objetivo que las de *HGA*-1.

VI.5.2. Comparaciones de las variantes del híbrido intercalado con micro algoritmo genético

Para el tercer híbrido (o *HGA*-3) es necesario utilizar un algoritmo con una población más pequeña, a este algoritmo se le llama micro algoritmo genético (*micro genetic algorithm*, MGA). El problema con este tipo de algoritmos es que tiende a quedar atrapado en un óptimo local de forma prematura, esto es en las generaciones iniciales del ciclo evolutivo. Para evitar la convergencia prematura se proponen distintos tipos de perturbación PERT, RAND, PERT-RAND y PCBAP (ver Sección V.6.3).

En esta sección se reportan los experimentos preliminares sobre distintos tipos de configuración del MGA, variando las probabilidades de mutación y recombinación, así como los métodos de terminación por generaciones, tiempo o ambos. Además, para todas las ejecuciones del MGA se agregó, la siguiente condición de terminación. El MGA se detiene si la población continúa estancada durante tres generaciones seguidas después de la generación 500. Esto se aplica ya que al examinar distintas ejecuciones se observó que la mejora después de la generación 500 es muy lenta. En la Tabla XXIII se observan las distintas configuraciónes del MGA con su respectiva abreviatura.

Tabla XXIII: Tipos de configuración para el micro algoritmo genético.

Abreviatura	Máximo número de generaciones	Máximo tiempo (ms)	Método de Terminación	Probabilidad de cruzamiento	Probabilidad de mutación
M	1000	–	Por generaciones	0.6	0.3
M2	3000	20000	Por generaciones o tiempo	0.6	0.3
M2*	450	20000	Por generaciones o tiempo	0.6	0.3
M3*	450	–	Por generaciones	0.9	0.5

Con las configuraciones de la Tabla XXIII se realizaron pruebas utilizando solamente un método de perturbación de entre *PCBAP, RAND, PERT, PERT-RAND*; esto para comprobar cuál tiene un mejor desempeño si se utiliza de forma independiente. También

se ejecutan pruebas utilizando tres métodos de perturbación y cambiando el orden en que se aplican, para asegurarse que el orden no influye o en el caso de tener algún efecto, tomar el que mejore más rapído la función objetivo de la población y ayude a escapar del óptimo local. El orden de aplicación se identifica con un acrónimo que se muestra en la Tabla XXIV.

Tabla XXIV: Acrónimos de los algoritmos genéticos híbridos cambiando el orden de los métodos de perturbación.

HGA-3_1	*HGA*-3_2	*HGA*-3_3	*HGA*-3_4	*HGA*-3_5	*HGA*-3_6
RAND	*RAND*	*PCBAP*	*PCBAP*	*PERT*	*PERT*
PCBAP	*PERT*	*RAND*	*PERT*	*PCBAP*	*RAND*
PERT	*PCBAP*	*PERT*	*RAND*	*RAND*	*PCBAP*

Para determinar qué configuraciones son las mejores y qué métodos de perturbación dan mejores resultados se realiza un análisis estadístico. La experimentación del MGA se realiza con 17 grafos de 51 vértices y con 30 ejecuciones por grafo. Dado que se realizan múltiples análisis estadísticos, se cuenta el número de ocasiones que tiene mejores resultados una variante con respecto a la otra y estás se ordenan de acuerdo a la mayor ganancia y al menor tiempo. Los resultados de la comparación de los métodos de perturbación se muestran en las tablas XXV y XXVI para las variantes por orden de ejecución y de ejecución independiente, respectivamente. Con la Tabla XXVI se puede concluir que el método de perturbación que genera mejores resultados es *PCBAP*; sin embargo, también es el que más tiempo tarda. Algo similar ocurre con los métodos de perturbación por orden, ya que en *HGA*-3_3 y *HGA*-3_4 se aplica primero *PCBAP* y son los que obtienen mayor ganancia en su función objetivo, pero a su vez los que también más tiempo consumen. Esto pasa porque al principio del ciclo evolutivo, *PCBAP* mejora mucho la función objetivo de la población en poco tiempo, es decir que si se aplica *PCBAP* en las primeras generaciones se avanza mucho en aptitud; por el

Tabla XXV: Mejor método de perturbación cambiando el orden de ejecución.

Mayor ganancia	Mejor tiempo
HGA-3_3	*HGA*-3_6
HGA-3_4	*HGA*-3_2
HGA-3_1	*HGA*-3_1
HGA-3_2	*HGA*-3_5
HGA-3_6	*HGA*-3_3
HGA-3_5	*HGA*-3_4

Tabla XXVI: Mejor método de perturbación al aplicarse de forma independiente.

Mayor ganancia	Mejor tiempo
PCBAP	*RAND*
PERT	*PERT_RAND*
PERT_RAND	*PERT*
RAND	*PCBAP*

contrario, si ya se tiene una población con una función objetivo alta, la mejora que propociona *PCBAP* es muy poca, o incluso nula, en caso de que la población esté atrapada en un óptimo local.

Con los mismos experimentos se define con cuál de las cuatro variantes de la Tabla XXIII el MGA obtiene mejores resultados. En las tablas XXVII y XXVIII se muestra que la configuración que siempre da mejores resultados es M, siendo también la que más tiempo tarda. Una vez que se define la mejor configuración con base en la ganancia de su función objetivo (M), así como el mejor método de perturbación de forma independiente (*PCBAP*) y de distinto orden (*HGA*-3_3), se realiza un análisis estadístico para compararlos entre sí; los resultados de este análisis se muestran en la Tabla XXIX. En dicha tabla, el símbolo "=" significa que no hubo diferencia significativa entre ambas muestras. Con base en los resultados de esta tabla se puede ver que siempre es más rápido aplicar los métodos de perturbación en conjunto con *HGA*-3_3 que *PCBAP* de

forma independiente. Con respecto a la ganancia se observa que en la mayoría de las ocasiones son iguales, es decir, no existe diferencia significativa, pero en algunos casos, concretamente para siete tipos de grafos, incluyendo el grafo completo, sí se genera una función objetivo mayor al aplicar solamente *PCBAP*.

Tabla XXVII: Mejor configuración del MGA al utilizar los métodos de perturbación con distinto orden.

Mayor ganancia	Mejor tiempo
M	M2*
M2	M3*
M2*	M2
M3*	M

Tabla XXVIII: Mejor configuración del MGA al utilizar los métodos de perturbación de forma independiente.

Mayor ganancia	Mejor tiempo
M	M2*
M3*	M2
M2*	M3*
M2	M

Finalmente, con los resultados preliminares de este capítulo se obtiene que los mejores operadores de variación para la representación NetKeys son la recombinación aritmética simple y la mutación por cambio de peso aplicada 5 veces. En el caso de la representación EdgeSet, la mutación que obtiene mejores resultados es la de borrado aleatorio de arista. Además, se determina que el operador de recombinación de Dandelion genera soluciones inválidas cuando su entrada no es un grafo completo. En el caso del algoritmo determinístico, se observa que el enfoque cooperativo genera mejores resultados que el enfoque no cooperativo, y que al utilizar *PrimRST* como árbol inicial, tiene mejor desempeño que con otros árboles iniciales. En el caso de los algorit-

Tabla XXIX: Resultado del análisis estadístico de las comparaciones entre el mejor método de perturbación independiente y el mejor orden al aplicarlos en conjunto.

Tipos de grafos	Mayor ganancia	Menor tiempo
barabasi 2*	PCBAP	*HGA*-3_3
barabasi 2	PCBAP	*HGA*-3_3
barabasi 1*	=	*HGA*-3_3
barabasi 1	=	*HGA*-3_3
completo	PCBAP	*HGA*-3_3
erdoreny 1*	=	*HGA*-3_3
erdoreny 1	PCBAP	*HGA*-3_3
erdoreny 2*	PCBAP	*HGA*-3_3
erdoreny 2	=	*HGA*-3_3
evolstoc *	=	*HGA*-3_3
evolstoc	=	*HGA*-3_3
geometri *	=	*HGA*-3_3
geometri	=	*HGA*-3_3
gradoady *	PCBAP	*HGA*-3_3
gradoady	=	*HGA*-3_3
watzstro	=	*HGA*-3_3
watzstro *	PCBAP	*HGA*-3_3

mos híbridos se concluye que las mejores variantes del algoritmo híbrido en cascada es *HGA*-1_1; para el híbrido intercalado que utiliza un MGA se selecciona *HGA*-3_3; y el para el híbrido con método de perturbación el *GA-PCBAP*.

Capítulo VII

Experimentación y análisis de resultados

En este capítulo se realizan las comparaciones finales para determinar la mejor representación para el algoritmo genético y el mejor algoritmo híbrido de las variantes propuestas en esta investigación. Una vez determinado el mejor algoritmo propuesto, éste se compara con el mejor algoritmo determinístico propuesto por Fajardo y Fernández (2010).

VII.1. Comparaciones entre representaciones

En las secciones VI.1 y VI.2 se definen los operadores evolutivos con los que se obtienen los mejores resultados para las representaciones NetKeys y EdgeSet, respectivamente. En esta sección se comparan estas dos representaciones para determinar cuál es la mejor. En la Tabla XXX se observa la comparación entre EdgeSet y NetKeys; en esta tabla se puede observar que EdgeSet es ampliamente superior tanto en promedio como en el valor máximo de la función objetivo, e incluso su tiempo de ejecución es menor. En la Figura 37 se observa un ejemplo de árbol solución que se obtiene al aplicar EdgeSet, este árbol es el óptimo para un grafo de 51 vértices, ya que tiene un valor de su función objetivo de 1900. En la Figura 38 se observa un ejemplo de árbol que se obtiene al aplicar NetKeys, el cual su función objetivo tiene un valor de 1854; en este par de figuras se muestra el grafo de entrada subyacente con aristas grises y las aristas del árbol solución que se genera con un color más oscuro, además, se muestra el color de los vértices y la etiqueta numérica de cada uno de ellos; para estos árboles, la raíz

Tabla XXX: Comparación entre las representaciones EdgeSet y NetKeys.

Tipos de grafos	EdgeSet			NetKeys		
	Función Objetivo		Tiempo (ms)	Función Objetivo		Tiempo (ms)
	Prom.	Max.	Max.	Prom.	Max.	Max.
barabasi 1	**1537.9**	**1577**	41209	1516.6	1559	55586
barabasi 2	**1608.9**	**1641**	41205	1579.9	1628	58687
erdoreny 1	**1790.1**	**1821**	38326	1711.5	1791	62525
erdoreny 2	**1820.4**	**1853**	37926	1732.4	1810	64298
evolstoc	**1773.6**	**1806**	39404	1689.0	1750	66597
geometri	**1625.5**	**1690**	43266	1537.9	1622	61044
gradoady	**1768.9**	**1856**	38702	1738.8	1826	68363
watzstro	**1779.5**	**1813**	37899	1701.9	1754	56938
erdoreny 1*	**1880.7**	**1900**	38482	1817.6	1872	63921

Tabla XXXI: Comparación entre las representaciones EdgeSet y Dandelion utilizando como entrada un grafo completo.

Tipo de grafo	EdgeSet					Dandelion				
	Función Objetivo			Tiempo (ms)		Función Objetivo			Tiempo (ms)	
	Prom.	Max.	Desv.	Prom.	Max.	Prom.	Max.	Desv.	Ave.	Max.
completo	1749.3	**1900**	81.2	78293.1	81875	**1809.8**	1875	35.2	45870.4	53323

es el vértice con etiqueta 0.

En la Sección VI.3 se determina que el método de recombinación de Dandelion tiende a generar árboles inválidos, por lo que solo se compara EdgeSet y Dandelion utilizando como entrada un grafo completo; esta comparación se observa en la Tabla XXXI. En esta comparación, EdgeSet genera una mayor ganancia al evaluar la función objetivo, el cual es el óptimo del grafo completo; sin embargo, para el valor de la función objetivo promedio y menor tiempo Dandelion tiene mejores resultados.

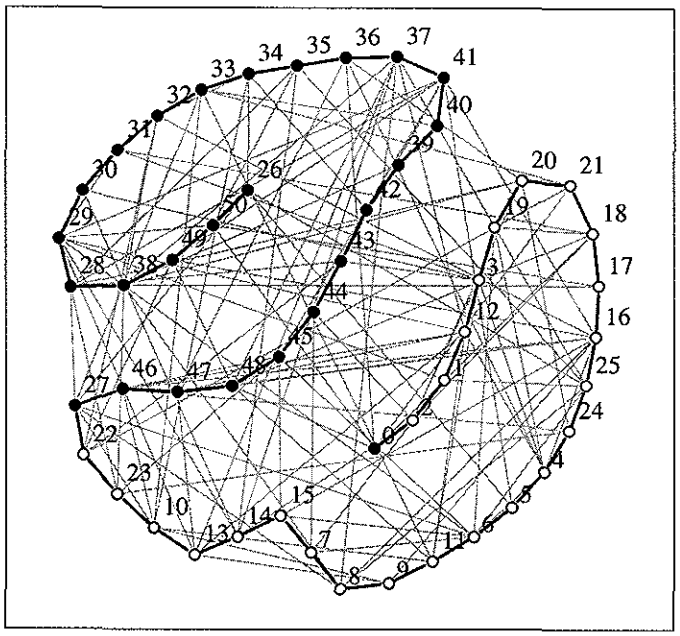


Figura 37: Árbol generado al usar el algoritmo genético con la representación EdgeSet en un grafo de 51 vértices.

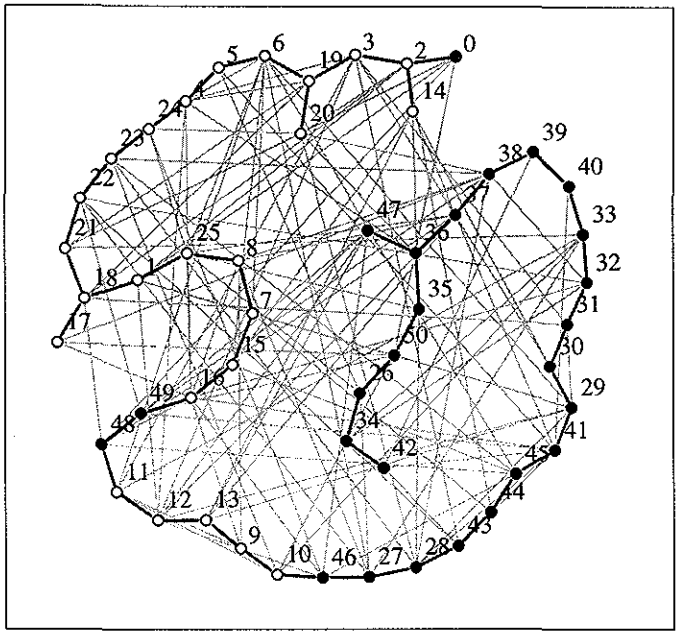


Figura 38: Árbol generado al usar el algoritmo genético con la representación NetKeys en un grafo de 51 vértices.

VII.2. Comparaciones entre EdgeSet y el algoritmo determinístico cooperativo

En la Sección VII.1 se determina que la representación que da mejores resultados, sin importar el grafo de entrada es EdgeSet. En esta sección se muestra la comparación que se realiza entre el algoritmo determinístico CBAP con enfoque cooperativo y el algoritmo genético con la represetación EdgeSet. Los tiempos de ejecución del algoritmo determinístico son muy pequeños comparados con EdgeSet, para realizar una comparación justa se realizan más ejecuciones del determinístico que del algoritmo genético. En la Tabla XXXII se observa que el tiempo de ejecución promedio del algoritmo determinístico es 9.27 milisegundos y del algoritmo genético 41222.5 milisegundos. Al redondear el valor de los promedios, se obtiene que la relación entre los tiempos de ejecución es $\frac{50000}{10}$, es decir, una ejecución del algoritmo genético equivale a correr 5000 veces el CBAP. La Tabla XXXIII muestra la comparación entre 5000 ejecuciones del algoritmo determinístico y el algoritmo genético con la representación EdgeSet. En esta tabla también se observa que EdgeSet obtiene mejores resultados al evaluar la función objetivo de las soluciones. Nótese que además en la Tabla XXXIII se contabiliza cuántas veces encuentra EdgeSet un equilibrio durante su ejecución, ya que el CBAP termina al encontrar un equilibrio.

Tabla XXXII: Comparación de los tiempos de ejecución entre el algoritmo genético EdgeSet y algoritmo determinístico con enfoque cooperativo.

Tipos de grafos	EdgeSet		CBAP coop	
	Prom.	Max.	Prom.	Max.
barabasi 2*	39799.7	41209	6.3	13
barabasi 1*	40501.4	41205	8.4	97
barabasi 1	41323.2	43472	4.3	16
barabasi 2	40345.9	41739	4.1	8
completo	78293.1	81875	47.2	99
erdoreny 1*	38625.7	39404	8.3	20
erdoreny 1	40008.9	41176	6.6	15
erdoreny 2*	40228.2	42476	7.5	19
erdoreny 2	42364.5	43266	7.2	17
evolstoc *	37852.2	38482	9.2	19
evolstoc	37697.6	38326	7.4	77
geometri *	36536.2	37926	6.4	19
geometri	38060.0	40472	5.1	17
gradoady *	36343.9	38131	6.8	18
gradoady	37556.8	38702	8.6	16
watzstro	37275.8	37899	5.2	11
watzstro *	37969.7	38781	9.1	16
Promedios:	41222.5		9.27	

Tabla XXXIII: Comparación entre el algoritmo genético EdgeSet y algoritmo determinístico con enfoque cooperativo con tiempos de ejecución similares.

Tipos de grafos	EdgeSet					CBAP Cooperativo		
	Función Objetivo			Equilibrios		Función Objetivo		
	Prom.	Max.	Desv.	Prom.	Max.	Prom.	Max.	Desv.
barabasi 2*	**1846.2**	**1892**	27.1	2.3	5	1681	1879	81.6
barabasi 1*	**1838.1**	**1882**	27.8	2.9	6	1684.4	1860	53.7
barabasi 1	**1537.9**	**1577**	13.7	2.2	5	1427.8	1538	25.6
barabasi 2	**1608.9**	**1641**	19.5	2.2	4	1490.6	1601	33.0
completo	**1749.3**	**1900**	81.2	0.2	2	1731.3	1892	58.1
erdoreny 1*	**1880.7**	**1900**	15.0	2.7	6	1720.8	1862	58.9
erdoreny 1	**1790.1**	**1821**	16.2	3.4	7	1614.4	1759	47.4
erdoreny 2*	**1786.7**	**1900**	16.2	3.4	7	1614.4	1759	47.4
erdoreny 2	**1820.4**	**1853**	23.0	2.5	8	1647.4	1817	53.6
evolstoc *	**1890.2**	**1900**	5.2	2.1	5	1787.0	1889	49.6
evolstoc	**1773.6**	**1806**	23.3	2.2	5	1590.0	1729	47.9
geometri *	**1803.3**	**1896**	49.0	4.6	7	1558.2	1798	65.3
geometri	**1625.5**	1690	43.1	4.1	10	1430.6	**1693**	50.1
gradoady *	**1797.7**	**1876**	74.9	1.5	6	1709.5	1862	56.4
gradoady	**1768.9**	**1856**	72.6	1.5	5	1652.1	1818	49.7
watzstro	**1779.5**	**1813**	27.7	2.7	8	1570.0	1769	52.1
watzstro *	**1862.6**	**1900**	52.4	2.7	8	1725.3	1872	55.6

VII.3. Comparación entre híbrido y algoritmo genético simple

En el caso de los algoritmos híbridos de la Sección VI.5 se determina que las mejores variantes del algoritmo híbrido en cascada es *HGA*-1_1; para el híbrido intercalado que utiliza un MGA se seleccionan las variantes *HGA*-3_3 de métodos de pertubación y el *GA-PCBAP* de forma independiente. Una comparación gráfica se muestra en la Figura 39; en esta figura se observa que, en general, la mayoría de los resultados tienden a estar cerca de 1900 (que es el valor óptimo) y algunos pocos son menores a 1800. La comparación utilizando análisis estadísticos de estas tres variantes se muestran en la Tabla XXXIV. De forma general se puede ver que en cuanto a los valores de la función objetivo y tiempo *PCBAP*> *HGA*-3_3 > *HGA*-1_1, es decir, *PCBAP* obtiene mayor ganancia pero es el más lento, la variante *HGA*-1_1 es la más rápida pero la que da peores resultados en la función objetivo de estas tres y la variante *HGA*-3_3 tiene resultados intermedios. El árbol óptimo con un valor de su función objetivo de 1900, resultado de aplicar *HGA*-1_1 a un grafo completo, se observa en la Figura 40; el grafo de entrada subyacente se observa por medio de aristas grises, mientras que el árbol solución generado es el que se muestra con aristas más gruesas.

VII.4. Comparación utilizando grafos de distintas densidades

En la mayoría de las comparaciones anteriores se utilizan grafos aleatorios con 51 vértices y con un número de aristas entre 130 y 190 (grafos dispersos), solamente una

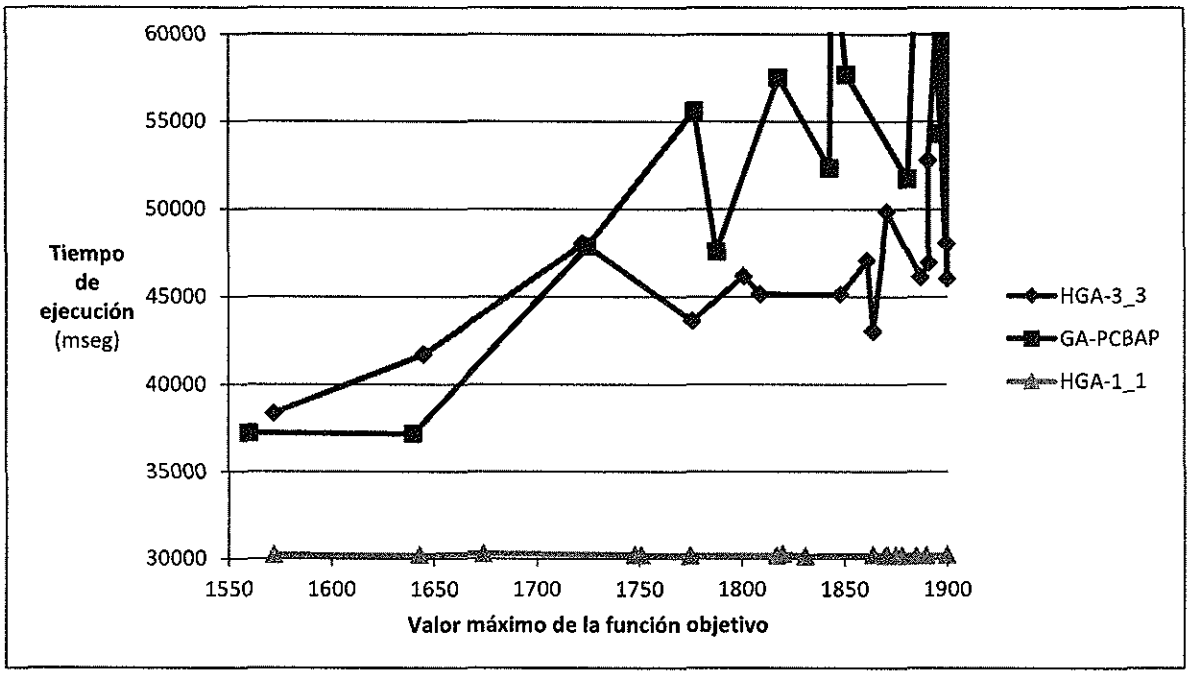


Figura 39: Valor máximo de la función objetivo vs tiempo de ejecución de las variantes del híbrido.

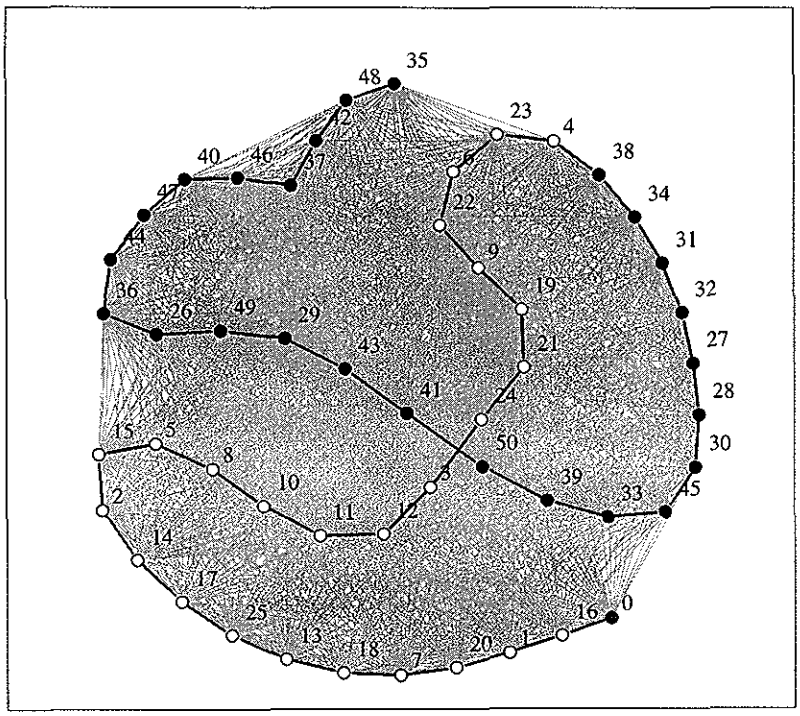


Figura 40: Árbol generado al aplicar el *HGA-1_1* en un grafo completo de 51 vértices.

Tabla XXXIV: Análisis estadístico de las comparaciones entre las variantes de los tres tipos de algoritmos híbridos.

Tipos de grafos	*HGA-3_3* vs *GA-PCBAP*		*HGA-1_1* vs *GA-PCBAP*		*HGA-1_1* vs *HGA-3_3*	
	Mayor ganancia	Menor tiempo	Mayor ganancia	Menor tiempo	Mayor ganancia	Menor tiempo
barabasi 2*	*GA-PCBAP*	*HGA-3_3*	*GA-PCBAP*	*HGA-1_1*	*HGA-3_3*	*HGA-1_1*
barabasi 2	*GA-PCBAP*	*HGA-3_3*	*GA-PCBAP*	*HGA-1_1*	=	*HGA-1_1*
barabasi 1*	=	*GA-PCBAP*	=	*HGA-1_1*	*HGA-3_3*	*HGA-1_1*
barabasi 1	=	*GA-PCBAP*	*GA-PCBAP*	*HGA-1_1*	=	*HGA-1_1*
completo	*GA-PCBAP*	*GA-PCBAP*	*GA-PCBAP*	*HGA-1_1*	*HGA-3_3*	*HGA-1_1*
erdoreny 1*	=	*GA-PCBAP*	*GA-PCBAP*	*HGA-1_1*	*HGA-3_3*	*HGA-1_1*
erdoreny 1	*GA-PCBAP*	*HGA-3_3*	*GA-PCBAP*	*HGA-1_1*	*HGA-3_3*	*HGA-1_1*
erdoreny 2*	*GA-PCBAP*	*HGA-3_3*	*GA-PCBAP*	*HGA-1_1*	*HGA-3_3*	*HGA-1_1*
erdoreny 2	=	*HGA-3_3*	*GA-PCBAP*	*HGA-1_1*	*HGA-3_3*	*HGA-1_1*
evolstoc *	=	*HGA-3_3*	*GA-PCBAP*	*HGA-1_1*	*HGA-3_3*	*HGA-1_1*
evolstoc	=	*HGA-3_3*	*GA-PCBAP*	*HGA-1_1*	*HGA-3_3*	*HGA-1_1*
geometri *	=	*HGA-3_3*	*GA-PCBAP*	*HGA-1_1*	*HGA-3_3*	*HGA-1_1*
geometri	=	*HGA-3_3*	*GA-PCBAP*	*HGA-1_1*	*HGA-3_3*	*HGA-1_1*
gradoady *	*GA-PCBAP*	*HGA-3_3*	*GA-PCBAP*	*HGA-1_1*	*HGA-3_3*	*HGA-1_1*
gradoady	=	*HGA-3_3*	*GA-PCBAP*	*HGA-1_1*	*HGA-3_3*	*HGA-1_1*
watzstro	=	*HGA-3_3*	*GA-PCBAP*	*HGA-1_1*	*HGA-3_3*	*HGA-1_1*
watzstro *	*GA-PCBAP*	*HGA-3_3*	*GA-PCBAP*	*HGA-1_1*	*HGA-3_3*	*HGA-1_1*

Tabla XXXV: Parámetros de grafos con distintas densidades utilizando el modelo Barabási-Albert.

Número total de aristas	Conexión preferencial	Atracción de los vértices sin aristas adyacentes (zero appeal)	Aristas a agregar en cada paso
128	1	1	4
256	1	1	15
512	0.85	1	30
1024	0.70	1	60
2048	0.45	1	800

Tabla XXXVI: Parámetros de grafos con distintas densidades utilizando el modelo evolutivo estocástico.

Número total de aristas	Número promedio de aristas	*citation*	Aristas a agregar en cada paso
128	124	*true*	2
256	230	*true*	4
512	508	*true*	10
1024	990	*true*	25
2048	1994	*true*	140

entrada es un grafo completo con 1275 aristas. Para dar mayor validez a los resultados en esta sección se utilizan grafos con 65 vértices y distintas densidades, siendo estas densidades de 128, 256, 512, 1024 y 2048 aristas. Para estos experimentos no se utiliza el grafo completo, el que para un grafo de 65 vértices tiene 2080 aristas. Con esto se asegura que las conclusiones que se obtengan no serán solamente para grafos aleatorios dispersos, si no para una variedad de grafos aleatorios de distintas densidades. Los modelos de grafos que se utilizan en esta etapa son el modelo Barabási-Albert (con los parámetros que se muestran en la Tabla XXXV y con la raíz con un alto grado de conectividad), el modelo evolutivo estocástico (con los parámetros que se muestran en la Tabla XXXVI), y el modelo Erdös Rényi con la variante $G(n, m)$ (en este modelo solo se recibe como parámetro el número de aristas exacto que se quiere generar).

Se realizan experimentos entre los dos mejores algoritmos híbridos, el *HGA*-1_1, que utiliza el orden 1_1, el *GA-PCBAP*, que utiliza la perturbación *PCBAP*, y el *GA*-

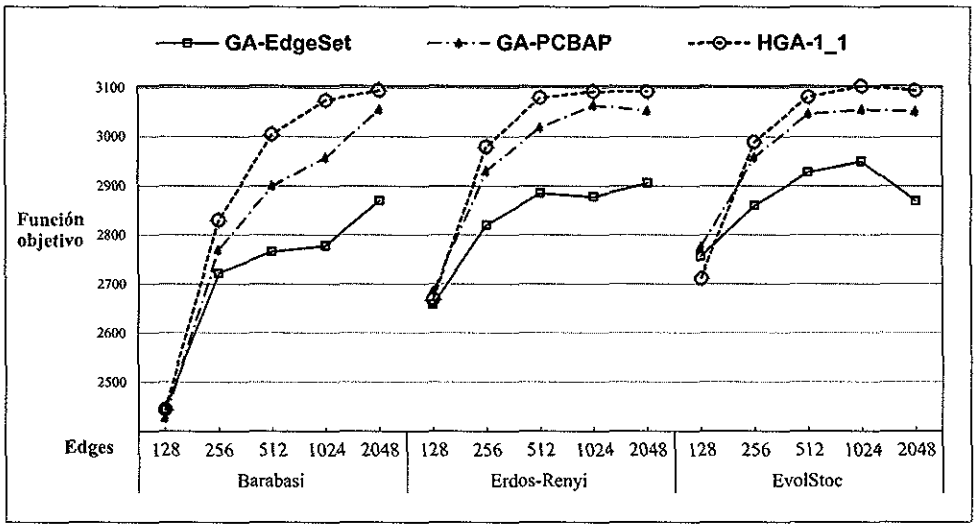


Figura 41: Gráfica comparativa de la máxima función objetivo de las mejores variantes de los algoritmos genéticos considerando como entrada tres modelos de grafos con distintas densidades.

ESet. que es el algoritmo genético sin método de perturbación que utiliza *EdgeSet.* La Figura 41 muestra una gráfica con la comparación del valor de la función objetivo de los algoritmos tomando en cuenta los tres tipos de grafos y las cinco densidades. En la Tabla XXXVII se muestran los resultados; en esta tabla se puede observar que los resultados para los grafos de 256, 512, 1024 y 2048 aristas son los mismos. Para estos casos, *HGA*-1_1 es el que genera mayor ganancia en menor tiempo, seguido de *GA-PCBAP*, el cual le gana al algoritmo genético simple *GA-ESet*; mientras que para los grafos de 128 aristas se mantiene *HGA*-1_1 siendo el que mejor resultado da y en menor tiempo pero con un menor margen. De la comparación entre *GA-ESet* y *GA-PCBAP*, se concluye que *GA-ESet* genera mejores resultados, pero *GA-PCBAP* utiliza menos tiempo.

Ya se sabe de antemano que los tiempos del algoritmo determinístico *CBAP* son mucho más rápidos que el genético, pero los resultados del algoritmo genético son mejores; lo que se quería lograr al implementar un algoritmo híbrido es reducir los tiempos

Tabla XXXVII: Análisis estadístico de las comparaciones entre los mejores algoritmos híbridos y el algoritmo genético utilizando como entrada grafos de distintas densidades.

Densidad	Grafos	*GA-ESet* vs *GA-PCBAP*		*GA-ESet* vs *HGA-1_1*		*GA-PCBAP* vs *HGA-1_1*	
		Mayor ganancia	Menor tiempo	Mayor ganancia	Menor tiempo	Mayor ganancia	Menor tiempo
128	barabasi 2	*GA-ESet*	*GA-PCBAP*	=	*HGA-1_1*	*HGA-1_1*	*GA-PCBAP*
	erdoreny 1	*GA-ESet*	*GA-PCBAP*	*HGA-1_1*	*HGA-1_1*	*HGA-1_1*	*HGA-1_1*
	evolstoc	*GA-ESet*	*GA-PCBAP*	*GA-ESet*	*HGA-1_1*	=	*HGA-1_1*
256	barabasi 2	*GA-PCBAP*	*GA-PCBAP*	*HGA-1_1*	*HGA-1_1*	*HGA-1_1*	*HGA-1_1*
	erdoreny 1	*GA-PCBAP*	*GA-PCBAP*	*HGA-1_1*	*HGA-1_1*	*HGA-1_1*	*HGA-1_1*
	evolstoc	*GA-PCBAP*	*GA-PCBAP*	*HGA-1_1*	*HGA-1_1*	*HGA-1_1*	*HGA-1_1*
512	barabasi 2	*GA-PCBAP*	*GA-PCBAP*	*HGA-1_1*	*HGA-1_1*	*HGA-1_1*	*HGA-1_1*
	erdoreny 1	*GA-PCBAP*	*GA-PCBAP*	*HGA-1_1*	*HGA-1_1*	*HGA-1_1*	*HGA-1_1*
	evolstoc	*GA-PCBAP*	*GA-PCBAP*	*HGA-1_1*	*HGA-1_1*	*HGA-1_1*	*HGA-1_1*
1024	barabasi 2	*GA-PCBAP*	*GA-PCBAP*	*HGA-1_1*	*HGA-1_1*	*HGA-1_1*	*HGA-1_1*
	erdoreny 1	*GA-PCBAP*	*GA-PCBAP*	*HGA-1_1*	*HGA-1_1*	*HGA-1_1*	*HGA-1_1*
	evolstoc	*GA-PCBAP*	*GA-PCBAP*	*HGA-1_1*	*HGA-1_1*	*HGA-1_1*	*HGA-1_1*
2048	barabasi 2	*GA-PCBAP*	*GA-PCBAP*	*HGA-1_1*	*HGA-1_1*	*HGA-1_1*	*HGA-1_1*
	erdoreny 1	*GA-PCBAP*	*GA-PCBAP*	*HGA-1_1*	*HGA-1_1*	*HGA-1_1*	*HGA-1_1*
	evolstoc	*GA-PCBAP*	*GA-PCBAP*	*HGA-1_1*	*HGA-1_1*	*HGA-1_1*	*HGA-1_1*

de ejecución y mejorar los resultados de la función objetivo, esto se logró. Una vez que se define que el mejor algoritmo genético es *HGA*-1_1, de entre las distintas representaciones, híbridos y métodos de pertubación, se hace una comparación utilizando grafos de distintas densidades con el algoritmo determinístico *CBAP*. Esta comparación se presenta en la Tabla XXXVIII. Como se observa, el *HGA*-1_1 supera en ganancia al algoritmo determinístico *CBAP* para todos los tipos de grafos, con todas las densidades; pero en tiempo el *CBAP* es mejor para todos los casos.

Tabla XXXVIII: Resultados del análisis estadístico de las comparaciones entre los mejores algoritmos híbridos y el algoritmo determinístico utilizando como entrada grafos de distintas densidades.

Densidad	Grafos	*HGA-1_1* vs *CBAP*	
		Mayor ganancia	Menor tiempo
128	barabasi 2	*HGA-1_1*	*CBAP*
	erdoreny 1	*HGA-1_1*	*CBAP*
	evolstoc	*HGA-1_1*	*CBAP*
256	barabasi 2	*HGA-1_1*	*CBAP*
	erdoreny 1	*HGA-1_1*	*CBAP*
	evolstoc	*HGA-1_1*	*CBAP*
512	barabasi 2	*HGA-1_1*	*CBAP*
	erdoreny 1	*HGA-1_1*	*CBAP*
	evolstoc	*HGA-1_1*	*CBAP*
1024	barabasi 2	*HGA-1_1*	*CBAP*
	erdoreny 1	*HGA-1_1*	*CBAP*
	evolstoc	*HGA-1_1*	*CBAP*
2048	barabasi 2	*HGA-1_1*	*CBAP*
	erdoreny 1	*HGA-1_1*	*CBAP*
	evolstoc	*HGA-1_1*	*CBAP*

En la Figura 42 se observa una gráfica de la diferencia en la función objetivo entre los dos algoritmos, considerando los tres modelos de grafos aleatorios y los grafos de distintas densidades.

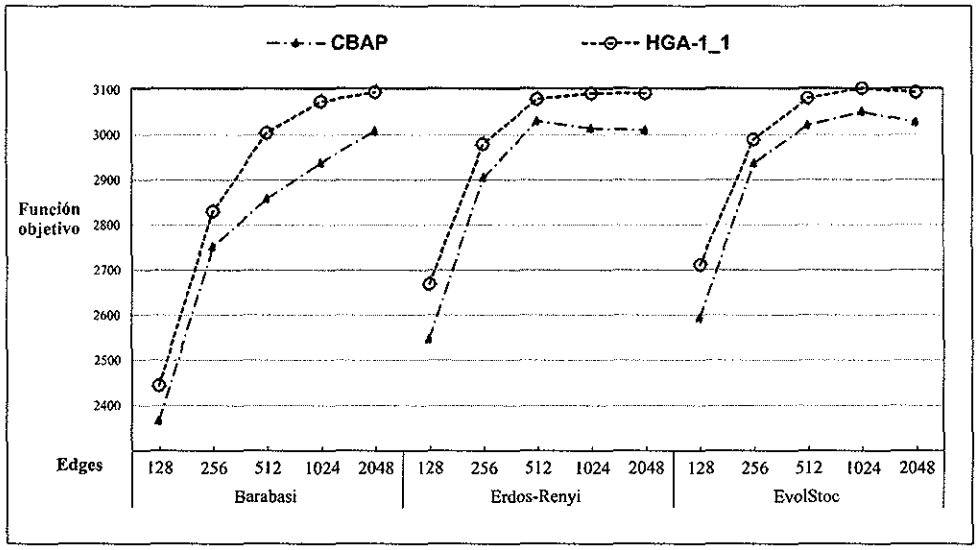


Figura 42: Gráfica comparativa de la máxima función objetivo de entre el algoritmo genético *HGA-1_1* y el determinístico *CBAP*.

Capítulo VIII

Conclusiones y trabajo futuro

En este capítulo se describen de forma general las conclusiones que se obtienen de la investigación realizada. También se proponen algunas variantes al PAG para que se vuelva un problema multiobjetivo y otros aspectos interesantes de cómo continuar esta investigación.

VIII.1. Conclusiones

El objetivo principal de esta tesis es el de mejorar los resultados que se obtienen con un algoritmo determinístico conocido para el PAG, para ello se consideran los algoritmos evolutivos como una buena opción para poder lograrlo. La técnica evolutiva que se selecciona son los algoritmos genéticos. Como parte de la experimentación preliminar se realizan comparaciones de representaciones para algoritmos genéticos con la finalidad de encontrar la representación con la que se obtengan mejores resultados para el PAG. Se determina que EdgeSet es la representación para algoritmos genéticos que obtiene mejores resultados para el PAG, sin importar el grafo aleatorio (tipo de entrada) ni la densidad de este. Otra representación que también genera buenas soluciones es Dandelion, pero esta tiende a obtener soluciones inválidas, dado el operador de recombinación que se utiliza. Además de experimentar con representaciones, se intenta disminuir el tiempo de ejecución utilizando algoritmos híbridos, los cuales combinan el algoritmo genético con el algoritmo determinístico; también se implementa un micro algoritmo genético para un híbrido intercalado, es decir, un algoritmo genético con poblaciones

muy pequeñas. Se determina que para el micro algoritmo genético es necesario utilizar métodos de perturbación; sin embargo, se demuestra empíricamente que el micro algoritmo genético no es una buena opción comparado con el híbrido en cascada. Finalmente, se define que la variante que mejores resultados genera, es la que se denomina *HGA*-1_1, la cual es un algoritmo híbrido en cascada que ejecuta primero el algoritmo determinístico CBAP con enfoque cooperativo y después el algoritmo genético con la representación EdgeSet. Finalmente se puede decir, que si se busca obtener una solución para el PAG sin importar la calidad de ésta, se recomienda utilizar el CBAP; más si se quiere encontrar una mejor solución, sin importar el tiempo de ejecución, se debe utilizar el algoritmo genético con la representación EdgeSet o un algoritmo híbrido que combine el algoritmo determinístico y el algoritmo genético.

VIII.2. Trabajo futuro

Un aspecto interesante con el que se puede continuar esta investigación es al evaluar los algoritmos evolutivos propuestos en esta tesis ante una variante multiobjetivo del PAG. En general para que un problema sea multiobjetivo tiene que intentar optimizar dos o más objetivos que estén en conflicto entre sí, por lo que una opción para un PAG multiobjetivo puede ser si se considera que cada vértice en lugar de tener solo una preferencia (color) contenga un vector de preferencias con base en los parámetros que se tengan que optimizar.

En la Sección III.3 se menciona que un óptimo global para un grafo completo es una cadena de vértices, por lo que un objetivo que puede entrar en conflicto con este tipo de estructura es el de minimizar la altura del árbol solución, es decir, sería por una parte la función objetivo tradicional y por otra una restricción que incentive a obtener

árboles con la menor altura posible.

Una técnica evolutiva que puede ser interesante probar con el PAG es la técnica de coevolución, la cual combina teoría de juegos con cómputo evolutivo. La técnica de coevolucion proviene de la observación biológica de que al coevolucionar un número de especies distintas, es más realista que simplemente evolucionar una población que contenga representantes de una sola especie, como sucede en los algoritmos genéticos. Los algoritmos coevolutivos funcionan de tal forma que en lugar de evolucionar una población de individuos similares que representen la solución global, coevolucionan subpoblaciones de individuos representantes de partes específicas de la solución global. Esto se podría enfocar para el PAG si se tienen dos poblaciones, una que optimice a los vértices blancos y otra a los vértices negros; para este caso, utilizando el punto de vista de teoría de juegos, cada población representa un jugador y lo que se busca sera llegar a un equilibrio de Nash. Existen trabajos recientes que aplican la técnica de coevolución para distintos tipos de problemas y utilizando multiples enfoques, Sun y Gong (2004), Son y Baldick (2004), Jiao *et al.* (2006), Hong y Jian (2006), Kim y Ryu (2007), Chang (2008), Goh y Tan (2009).

Referencias blibiográficas

Barabasi, A. L. y Albert, R. (1999). Emergence of Scaling in Random Networks. *Science*, **286**(5439): 509–512.

Beni, G. y Wang, J. (1993). Swarm intelligence in cellular robotic systems. En: P. Dario, G. Sandini, y P. Aebischer (eds.), *Robots and Biological Systems: Towards a New Bionics?*, Vol. 102 de *NATO ASI Series*, pp. 703–712. Springer Berlin Heidelberg.

Blum, C. y Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, **35**: 268–308.

Box, J. F. (1987). Guinness, gosset, fisher, and small samples. *Statistical Science*, **2**(1): 45–52.

Broder, A. (1989). Generating random spanning trees. En: *30th Annual Symposium on Foundations of Computer Science, 1989.*, pp. 442–447, 30 Octubre, North Carolina, USA.

Bui, T. N. y Zrncic, C. M. (2006). An ant-based algorithm for finding degree-constrained minimum spanning tree. En: *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, GECCO '06, pp. 11–18, 8-12 Julio, New York, USA. ACM.

Caminiti, S. y Petreschi, R. (2009). Parallel algorithms for dandelion-like codes. En: *Proceedings of the 9th International Conference on Computational Science: Part I*, ICCS '09, pp. 611–620, 25-27 Mayo, Louisiana, USA. Springer-Verlag.

Caminiti, S. y Petreschi, R. (2010). Parallel algorithms for encoding and decoding blob code. *WALCOM: Algorithms and Computation*, **5942**: 167–178.

Cayley, A. (1889). A theorem on trees. *Quartely Journal of Mathematics*, **23**: 376–378.

Chang, H. S. (2008). Converging coevolutionary algorithm for two-person zero-sum discounted markov games with perfect information. *IEEE Transactions on Automatic Control*, **53**(2): 596 –601.

Coello, C. A. C. y Pulido, G. T. (2001). A micro-genetic algorithm for multiobjective optimization. En: *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization*, EMO '01, pp. 126–140, 7-9 Marzo, Zurich, Switzerland. Springer-Verlag.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., y Stein, C. (2001). *Introduction to Algorithms*. Second edition. MIT Press. Massachusetts, USA, p. 1180. ISBN 9780262033848.

Csardi, G. y Nepusz, T. (2006). The igraph software package for complex network research. *InterJournal*, **Complex Systems**: 1695.

Dasgupta, A., Ghosh, S., y Tixeuil, S. (2006). Self-stabilization. En: *Proceedings of the 8th international conference on Stabilization, safety, and security of distributed systems*, SSS'06, pp. 231–243, 17-19 Noviembre, Dallas, TX, USA. Springer-Verlag.

Eiben, A. y Smith, J. (2003). *Introduction to evolutionary computing*. Natural computing series. Springer. Heidelberg, Germany, p. 299.

Erdös, P. y Rényi, A. (1959). On random graphs, I. *Publicationes Mathematicae (Debrecen)*, **6**: 290–297.

Fajardo, D. y Fernández, J. A. (2010). the bodyguard allocation problem. *(Manuscrito)*.

Farmer, J. D., Packard, N. H., y Perelson, A. S. (1986). The immune system, adaptation, and machine learning. *Phys. D*, **2**(1-3): 187–204.

Gen, M. y Cheng, R. (1997). *Genetic algorithms and engineering design*. Wiley series in engineering design and automation, first edition. Wiley-Interscience. p. 411.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.*, **13**: 533–549.

Glover, F. y Kochenberger, G. (2003). *Handbook of metaheuristics*. International series in operations research & management science. Kluwer Academic Publishers. Massachusetts, USA, p. 556.

Goh, C.-K. y Tan, K. C. (2009). A competitive-cooperative coevolutionary paradigm for dynamic multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, **13**(1): 103–127.

Goldberg, D. E. (1989a). Genetic algorithms and walsh functions: Part i, a gentle introduction. *Complex Systems*, **3**: 129–152.

Goldberg, D. E. (1989b). Genetic algorithms and walsh functions: Part ii, a gentle introduction. *Complex Systems*, **3**: 153–171.

Gottlieb, J., Julstrom, B. A., y Raidl, G. R. (2001). Prüfer numbers: A poor representation of spanning trees for evolutionary search. En: *In Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2001*, pp. 343–350, 7-11 Julio, San Francisco, California, USA. Morgan Kaufmann.

Holland, J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. Bradford Books. MIT Press. Cambridge, MA, USA, p. 211.

Hong, Z. y Jian, W. (2006). A cooperative coevolutionary algorithm with application to job shop scheduling problem. En: *Service Operations and Logistics, and Informatics, 2006. SOLI '06. IEEE International Conference on*, pp. 746–751, 21-23 Junio, Shanghai, CH.

Jiao, L., Liu, J., y Zhong, W. (2006). An organizational coevolutionary algorithm for classification. *IEEE Transactions on Evolutionary Computation*, **10**(1): 67–80.

Julstrom, B. A. (2001). The blob code: A better string coding of spanning trees for evolutionary search. En: *Genetic and Evolutionary Computation Conference Workshop Program*, pp. 256–261, 7-11 Julio, San Francisco, California, USA. Morgan Kaufmann.

Julstrom, B. A. (2004). Codings and operators in two genetic algorithms for the leaf-constrained minimum spanning tree problem. *International Journal of Applied Mathematics and Computer Science*, **14**(3): 385–396.

Julstrom, B. A. (2005). The blob code is competitive with edge-sets in genetic algorithms for the minimum routing cost spanning tree problem. En: *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pp. 585–590, 25-29 Junio, New York, NY, USA. ACM.

Kim, M. W. y Ryu, J. W. (2007). An efficient coevolutionary algorithm using dynamic species control. En: *Proceedings of the Third International Conference on Natural Computation - Volume 03*, ICNC '07, pp. 431–435, 24-27 Agosto, Washington, DC, USA. IEEE Computer Society.

Knowles, J. y Corne, D. (2000). A new evolutionary approach to the degree-constrained minimum spanning tree problem. *IEEE Transactions on Evolutionary Computation*, 4(2): 125–134.

Knowles, J. y Corne, D. (2001a). A comparison of encodings and algorithms for multiobjective minimum spanning tree problems. En: *Proceedings of the 2001 Congress on Evolutionary Computation (CEC01)*, Vol. 1, pp. 544–551, 27-30 Mayo, Seoul, Korea. IEEE Press.

Knowles, J. D. y Corne, D. W. (2001b). Benchmark problem generators and results for the multiobjective degree-constrained minimum spanning tree problem. En: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2001*, pp. 424–431, 7-11 Julio, San Francisco, California, USA. Morgan Kaufmann.

Krishnakumar, K. (1989). Micro-genetic algorithms for stationary and non-stationary function optimization. En: *Intelligent Control and Adaptive Systems, Proc. of the SPIE*, Vol. 1196, pp. 289–296.

Kumar, R., Raghavan, P., Rajagopalan, S., Sivakumar, D., Tomkins, A., y Upfal, E. (2000). Stochastic models for the web graph. En: *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pp. 57–65, 12-14 Noviembre, Washington, DC, USA. IEEE Computer Society.

Kumar, R., Singh, P., y Chakrabarti, P. (2005). Multiobjective ea approach for improved quality of solutions for spanning tree problem. En: C. Coello Coello, A. Hernández Aguirre, y E. Zitzler (eds.), *Evolutionary Multi-Criterion Optimization*, Vol. 3410 de *Lecture Notes in Computer Science*, pp. 811–825. Springer Berlin / Heidelberg.

Lilliefors, H. W. (1967). On the kolmogorov-smirnov test for normality with mean and variance unknown. *Journal of the American Statistical Association*, **62**(318): 399–402.

Moscato, P. (1999). *Memetic algorithms: a short introduction*, pp. 219–234. McGraw-Hill Ltd., UK, Maidenhead, UK, England.

Neumann, J., Morgenstern, O., Rubinstein, A., y Kuhn, H. (2007). *Theory of games and economic behavior*. Princeton Classic Editions. Princeton University Press. Princeton, NJ, USA, p. 739.

Nisan, N. (2007). *Algorithmic game theory*. Cambridge University Press. New York, NY, USA, p. 754.

Nummela, J. y Julstrom, B. A. (2006). An effective genetic algorithm for the minimum-label spanning tree problem. En: *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, GECCO '06, pp. 553–558, 8-12 Julio, New York, USA. ACM.

Palmer, C. y Kershenbaum, A. (1994). Representing trees in genetic algorithms. En: *IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on Evolutionary Computation*, Vol. 1, pp. 379–384, 27-29 Junio, New York, NY, USA.

Palmer, C. C. y Kershenbaum, A. (1995). An approach to a problem in network design using genetic algorithms. *Networks*, **26**(3): 151–163.

Paulden, T. y Smith, D. (2004). The rainbow code: A superior genetic algorithm representation for layered trees. En: *Proceedings of the 34th International Conference on Computers and Industrial Engineering*, Vol. 4, pp. 596–601, 14-16 Noviembre, San Francisco, CA, USA.

Paulden, T. y Smith, D. (2006a). From the dandelion code to the rainbow code: A class of bijective spanning tree representations with linear complexity and bounded locality. *IEEE Transactions on Evolutionary Computation*, **10**(2): 108–123.

Paulden, T. y Smith, D. (2006b). Recent advances in the study of the dandelion code, happy code, and blob code spanning tree representations. En: *IEEE Congress on Evolutionary Computation, 2006. CEC 2006.*, pp. 2111–2118, 16-21 Julio, Vancouver, Canada. IEEE.

Picciotto, S. (1999). *How to encode a tree*. University of California, San Diego. San Diego, CA, USA, p. 250.

Pisanski, T. y Randić, M. (1998). *Bridges between geometry and graph theory*. Institute of Mathematics, Physics and Mechanics, Department of Mathematics, University of Ljubljana. Univ. of Ljubljana, Inst. of Mathematics, Physics and Mechanics, Dep. of Mathematics.

Prüfer, H. (1918). Neuer beweis eines satzes über permutationen. *Arch. Math. Phys*, **27**: 742–744.

Raidl, G. R. y Julstrom, B. A. (2003). Edge sets: an effective evolutionary coding of spanning trees. *Evolutionary Computation, IEEE Transactions on*, **7**(3): 225–239.

Reynolds, R. G. (1994). An introduction to cultural algorithms. En: *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, pp. 131—139, 24-26 Febrero, San Diego, California, USA. World Scienfific Publishing.

Rothlauf, F. y Gaube, T. (2001). The link and node biased encoding revisited: Bias and adjustment of parameters. En: *Proceedings of the EvoWorkshops on Applications of Evolutionary Computing*, pp. 1–10, 18-20 Abril, London, UK, UK. Springer-Verlag.

Rothlauf, F. y Goldberg, D. E. (2003). Redundant representations in evolutionary computation. *Evolutionary Computation*, **11**: 381–415.

Rothlauf, F., Goldberg, D. E., y Heinzl, A. (2002). Network random keys - a tree representation scheme for genetic and evolutionary algorithms. *Evolutionary computation*, **10**(1): 75–97.

Schollmeier, R. (2001). A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. En: *Proceedings. First International Conference on Peer-to-Peer Computing, 2001.*, pp. 101–102, 27-29 Agosto, Linkoping, Sweden.

Seredynski, F. (2006). Evolutionary paradigms. En: A. Zomaya (ed.), *Handbook of Nature-Inspired and Innovative Computing*, pp. 111–145. Springer US.

Snedecor, G. y Cochran, W. (1989). *Statistical methods*. Número v. 276 en: Statistical Methods. Iowa State University Press. p. 503.

Son, Y. S. y Baldick, R. (2004). Hybrid coevolutionary programming for nash equilibrium search in games with local optima. *IEEE Transactions on Evolutionary Computation*, **8**(4): 305–315.

Storn, R. y Price, K. (1997). Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, **11**: 341–359.

Sun, X. y Gong, D. (2004). Multi-robot moving path planning based on coevolutionary algorithm. En: *Fifth World Congress on Intelligent Control and Automation, (WCICA 2004).*, Vol. 3, pp. 2231–2235, 14-18 Junio, Hangzhou, China.

Thompson, E., Paulden, T., y Smith, D. K. (2007). The dandelion code: A new coding of spanning trees for genetic algorithms. *IEEE Transactions on Evolutionary Computation*, **11**(1): 91–100.

Viger, F. y Latapy, M. (2005). Efficient and simple generation of random simple connected graphs with prescribed degree sequence. En: *in The Eleventh International Computing and Combinatorics Conference, (COCOON 2005)*, pp. 440–449, 16-29 Agosto, Kunming, China. Springer.

Vose, M. D. y Wright, A. H. (1998). The simple genetic algorithm and the walsh transform: Part i, theory. *Evolutionary Computation*, **6**: 253–273.

Wang, J. y Kusiak, A. (2001). *Computational intelligence in manufacturing handbook*. Advanced topics in mechanical engineering series. CRC Press. p. 576.

Watts, D. J. y Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. *Nature*, **393**(6684): 440–442.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin*, **1**(6): 80–83.

Wolpert, D. H. y Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, **1**(1): 67–82.

Wolpert, D. H. y Macready, W. G. (2005). Coevolutionary free lunches. *IEEE Transactions on Evolutionary Computation*, **9**(6): 721–735.

Yang, B. (2006). A hybrid evolutionary algorithm for the euclidean steiner tree problem using local searches. En: B. Gabrys, R. Howlett, y L. Jain (eds.), *Knowledge-Based Intelligent Information and Engineering Systems*, Vol. 4251 de *Lecture Notes in Computer Science*, pp. 60–67. Springer Berlin / Heidelberg.

Apéndice A

Procedimiento del análisis estadísticos

En este apéndice se describe la metodología que se sigue para los análisis estadísticos que se realizan en los Capítulos VI y VII. El análisis estadístico comprueba de manera formal que exista una diferencia significativa entre los resultados de dos muestras y si existe alguna diferencia se determina qué muestra es mayor. Al realizar estas pruebas se plantean dos hipótesis y se obtiene un valor p (*p-value*), el cual es la probabilidad de obtener un valor al menos tan extremo como el que se ha obtenido, suponiendo que la hipótesis nula es cierta. La hipótesis nula se rechaza si el valor p asociado al resultado observado es igual o menor que el nivel de significancia establecido, regularmente 0.05. Por lo tanto, solo se pueden obtener dos resultados, que se rechaze la hipótesis nula o que la hipótesis nula no se pueda rechazar con ese nivel de significancia (que no implica que la hipótesis nula sea verdadera). A continuación se explican todas las pruebas realizadas para el análisis.

A.1. Pruebas de normalidad

En esta sección se explica la prueba de normalidad utilizada para el análisis estadístico. Existen muchos tipos de pruebas para comprobar si un conjunto de datos se comporta como una distribución normal. Kolmogorov-Smirnov es una de las más utilizadas. La prueba Kolmogorov-Smirnov es una prueba no paramétrica que se utiliza para determinar la bondad de ajuste de dos distribuciones de probabilidad entre sí. En el caso que se requiera verificar la normalidad de una distribución, la prueba Lilliefors

(Lilliefors, 1967) conlleva algunas mejoras con respecto a la de Kolmogorov-Smirnov.

En estadística, la prueba de Kolmogorov-Smirnov es una prueba no paramétrica de la igualdad de las distribuciones continuas; se puede utilizar para comparar una muestra con una distribución de probabilidad de referencia o para comparar dos muestras. El estadístico de Kolmogorov-Smirnov cuantifica la distancia entre la función de distribución empírica de la muestra y la función de distribución acumulada de la distribución de referencia, o entre las funciones de distribución empírica de dos muestras.

La prueba de Kolmogorov-Smirnov se pueden modificar para servir como una prueba de bondad de ajuste, es decir, el grado de ajuste que existe entre la distribución obtenida a partir de la muestra y la distribución teórica que se supone debe seguir esa muestra. En el caso especial de la prueba de normalidad de la distribución, las muestras están estandarizadas y se compara con una distribución normal estándar. Esto equivale a establecer la media y la varianza de la distribución de referencia igual a las estimaciones de la muestra. Esta prueba se basa en la hipótesis nula de que no hay diferencias significativas entre la distribución muestral y la teórica.

En estadística, la prueba Lilliefors, llamada asi por Hubert Lilliefors, profesor de estadística de la universidad de Washington, es una adaptación de la prueba Kolmogorov-Smirnov. Se utiliza para probar la hipótesis nula de que la muestra de datos proviene de una población distribuida normalmente. Esta última prueba es la que se utiliza en esta investigación.

Para esta prueba se tienen dos hipótesis:

- **H0:** (nula) Los datos siguen una distribución normal.
- **H1:** (alternativa) Los datos no siguen una distribución normal.

A.2. Prueba para determinar igualdad de varianzas

Se denomina prueba-F (*F-test*) o prueba de Fisher en honor a Ronald A. Fisher (Snedecor y Cochran, 1989). En estadística aplicada, la prueba-F se utiliza para algunas hipótesis como, la hipótesis de que las medias de múltiples poblaciones normalmente distribuidas y con la misma desviación estándar son iguales o la hipótesis de que las desviaciones estándar de dos poblaciones normalmente distribuidas son iguales.

En esta investigación la prueba-F se puede utilizar para comprobar la hipótesis de que las desviaciones estándar de dos muestras son iguales. Las dos hipótesis de esta prueba son:

- **H0:** (nula) Las desviaciones estándar de las dos poblaciones normalmente distribuidas son iguales.
- **H1:** (alternativa) Las desviaciones estándar de las dos poblaciones normalmente distribuidas no son iguales.

A.3. Prueba t-Student

La prueba *t-Student* se introdujo en 1908 por William Sealy Gosset ('Student' era su seudónimo) (Box, 1987). Una prueba t es cualquier prueba de hipótesis estadística en la que la estadística de prueba sigue una distribución t de Student si la hipótesis nula es compatible. Existen muchas variantes de la prueba *t-Student*. Esta prueba se utiliza para una muestra para determinar si la media de la muestra coincide con una población distribuida normalmente, para dos muestras para determinar si la media de ambas muestras son iguales. Estas pruebas se suelen llamar *t-Student*, aunque en sentido estricto que el nombre sólo debe usarse si las varianzas de las dos muestras

se supone que sean iguales. La prueba utilizada con el supuesto de varianzas distintas se llama *t-Welch*. Estas pruebas se refieren a menudo como 'sin pareja' o 'muestras independientes', ya que generalmente se aplican cuando los datos de las dos muestras que se comparan no se traslapan. Otra variante, es cuando se supone que la diferencia entre las medias de las dos muestras es cero. Por ejemplo, supóngase que se mide el tamaño del tumor de un paciente con cáncer antes y después de un tratamiento. Si el tratamiento es eficaz, se espera que el tamaño del tumor para muchos de los pacientes sean más pequeños después del tratamiento. Esto se refiere a menudo como la 'pareja' o 'medidas repetidas' *t-test* apareados prueba de diferencia.

Un requisito para aplicar la *t-Student* es que las muestras sean normales. La prueba de *t-Student* se puede realizar para muestras independientes o dependientes (apareadas); para muestras con varianzas similiares o varianzas distintas y muestras de tamaños iguales o diferentes. Las dos hipótesis de esta prueba son:

- **H0:** (nula) Los datos normalmente distribuidos no muestran diferencia significativa, son iguales.
- **H1:** (alternativa) Los datos normalmente distribuidos muestran diferencia significativa, son diferentes.

A.4. Prueba de los signos de Wilcoxon

La prueba de los signos de Wilcoxon (*Wilcoxon Signed-rank test*) es una prueba no paramétrica para comparar la mediana de dos muestras relacionadas y determinar si existen diferencia entre ellas. Se utiliza como alternativa a la prueba t-Student cuando no se puede suponer la normalidad de los datos (Wilcoxon, 1945). Para esta prueba no

se presupone ningún tipo de distribución particular.

Para esta prueba se tienen dos hipótesis:

- **H0:** (nula) Los datos no muestran diferencia significativa, son iguales.
- **H1:** (alternativa) Los datos muestran diferencia significativa, son diferentes.

A.5. Algoritmo de análisis estadístico

Para realizar el análisis de manera automatica se utilizó el lenguaje de programación para análisis estadístico R. Todos los análisis realizados en esta investigación se realizan con muestras de 30 datos, ya que son los mínimos requeridos para realizar algunas pruebas.

En el Algoritmo 10 se utilizan las pruebas descritas anteriormente en forma de métodos, donde se regresa solamente el valor p; si el valor p es mayor que 0.05, no se rechaza la hipótesis nula de la prueba correspondiente. En la Línea 1 se utiliza el análisis Lilliefors para evaluar la normalidad de las muestras. En la Línea 2 se utiliza la prueba-F para evaluar la similitud entre las varianzas. En las Líneas 3 y 6 se utiliza la prueba t-Student de 2 colas (determina si existe diferencia significativa entre las muestras) y de 1 cola (determina cuál es la muestra mayor) para varianzas similares, respectivamente. Las mismas pruebas pero para varianzas distintas se aplica en las Líneas 13 y 16. Si las muestras no son normales se aplica la prueba de Wilcoxon en las Líneas 24 y 27, de 2 y 1 cola, respectivamente.

Algoritmo 10: Metodología del análisis estadístico.

Input : Dos muestras en los vectores números x, y.
Output : Una cadena indicando la muestra mayor 'x', 'y' o '=' si son iguales.

```
1  if lillieFors(x) > 0.05 and lillieFors(y) > 0.05 then
       ▷ Las muestras pertenecen a una distribución normal
2      if f_Test(x,y) > 0.05 then
           ▷ Las varianzas son similares
3          if tStudent_2Colas_varianzasIguales(x,y) > 0.05 then
               ▷ Los datos no muestran diferencia significativa
4              return " = ";
5          else
               ▷ Los datos muestran diferencia significativa
6              if tStudent_1Cola_varianzasIguales(x,y) > 0.05 then
7                  return "x";
8              else
9                  return "y";
10             end
11         end
12     else
           ▷ Las varianzas son distintas
13         if tStudent_2Colas_varianzasDiferentes(x,y) > 0.05 then
14             return " = ";
15         else
16             if tStudent_1Cola_varianzasDiferentes(x,y) > 0.05 then
17                 return "x";
18             else
19                 return "y";
20             end
21         end
22     end
23 else
       ▷ Las muestras no pertencen a una distribución normal
24     if wilcoxon_2Colas(x,y) > 0.05 then
25         return " = ";
26     else
27         if wilcoxon_1Cola(x,y) > 0.05 then
28             return "x";
29         else
30             return "y";
31         end
32     end
33 end
```