UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS

Estudio del problema Min-SA y algunas variantes: Algoritmos y heurísticas.

Por: Claudio Sebastián Álvarez Bernal

Tesis presentada a la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Concepción para optar al título profesional de Ingeniería Civil Matemática

Marzo 2025
Concepción, Chile

Profesor Guía: Christopher Thraves Caro
Profesor Co-Guía: Mauricio Soto Gómez

Abstract

Los grafos con signos son grafos en los que cada arista se asigna un signo positivo o negativo. Estos grafos se utilizan para representar diversas estructuras, siendo una de las aplicaciones más comunes el modelado de redes sociales o relaciones sociales. En este tipo de representación, las aristas con signos positivos representan una buena relación entre los vértices que conectan, mientras que las aristas con signos negativos representan una mala relación.

Kermarrec y Thraves en [16] plantearon el problema: dado un grafo con signos, ¿existe un embedding en el espacio métrico de la línea euclidiana tal que cada vértice esté más cerca de todos los vértices con los que tiene una buena relación (vértices unidos por aristas positivas) que de aquellos con los que tiene una mala relación (vértices unidos por aristas negativas)? A este problema se le conoce con el nombre de problema de Sitting Arrangement (SA), el cual ha sido ampliamente estudiado.

Pardo, Soto y Thraves, en [25], motivados por el problema SA, buscaron una solución para la versión de optimización del problema SA en la recta euclidiana. Así, se planteó el problema MinSA, que consiste en minimizar el número de errores producidos en un embedding en el espacio métrico de la línea euclidiana. Este problema ha sido abordado mediante enfoques teóricos y heurísticos. Una heurística, que resuelve con buenos resultados el problema MinSA, es la heurística BVNS.

En esta tesis, un objetivo es mejorar los resultados del BVNS implementando algoritmos genéticos combinados con la técnica estadística t-Distributed Stochastic Neighbor Embedding (t-SNE). Por medio de experimentos, se busca encontrar los mejores parámetros tanto para el algoritmo genético como para el t-SNE, para luego, comparar el rendimiento del algoritmo obtenido con el BVNS. Además, otro objetivo de esta tesis, es implementar tanto el algoritmo genético obtenido como el BVNS en dos problemas planteados en esta tesis, los cuales nacen del problema SA.

Índice general

Introducción

En los últimos años, los problemas de disposición de grafos han ganado relevancia debido a su importancia en el diseño de circuitos de Integración a Gran Escala (VLSI). Estos problemas se centran en cómo proyectar un grafo en otro predefinido, optimizando la disposición para minimizar aspectos como la dilatación, la congestión y la carga. Esta optimización es crucial para mejorar la eficiencia en la construcción y enrutamiento de circuitos [26].

Dentro de los problemas de disposición de grafos, la disposición lineal es la más estudiada. Sin embargo, también existen otras incrustaciones importantes, como en ciclos [20, 28, 29], grillas [2, 18, 22] y estructuras más complejas como árboles [33, 7, 13, 14], toros [15, 32] e hipercubos [17, 19]. Estas variantes permiten proyectar grafos en diferentes estructuras anfitrionas, adaptando la disposición según la geometría del anfitrión.

Los problemas de disposición de grafos tienen aplicaciones en diversas áreas además de diseño de circuitos de Integración a Gran Escala (VLSI). Por ejemplo, planificación de rutas y logística, los grafos ayudan a encontrar el camino más corto entre dos puntos, optimizando rutas de transporte [3]. En visualización de datos y redes sociales, problemas como Sentarse Más Cerca de Amigos que de Enemigos (SCFE) facilitan la comprensión de estructuras sociales complejas [16]. Además, el coloreo de grafos se aplica en la asignación de frecuencias en redes de comunicación y en problemas de scheduling [10]. La detección de anomalías utiliza grafos para capturar conexiones en tiempo real y detectar comportamientos anómalos en redes financieras o sociales [21]. Estas aplicaciones demuestran la versatilidad de la teoría de grafos en resolver problemas complejos en diversas

áreas.

El problema de *Sentarse Más Cerca de Amigos que de Enemigos (SCFE)* [16], más tarde renombrado como *problema Sitting Arrangement (SA)* [25], es un tipo de problema de disposición lineal con aplicaciones que van desde la organización social hasta la visualización de datos. Este problema consiste en ordenar los vértices de un grafo etiquetado con signos de manera que los vértices conectados por aristas positivas (amigos) estén más cerca entre sí, mientras que los vértices conectados por aristas negativas (enemigos) se mantengan alejados. La complejidad del problema radica en su naturaleza combinatoria, lo que hace que encontrar un orden óptimo sea difícil.

Debido a la dificultad de garantizar una solución óptima, se suelen buscar soluciones aproximadas que, aunque no siempre son óptimas, satisfacen razonablemente los criterios del problema. Este proyecto se centra en desarrollar soluciones aproximadas para tres variantes específicas del problema SA, explorando diferentes perspectivas y desafíos en optimización derivados del problema original. Una de estas variantes ya ha sido estudiada anteriormente, por lo cual se busca presentar una solución que mejore los resultados.

Definiciones y presentación del problema

En este capítulo, primero se presentarán los conceptos y definiciones necesarias para el desarrollo de esta tesis. Posteriormente, se planteará el problema a estudiar de interés de esta tesis.

2.1. Definiciones

Un ***grafo*** es un par $G = (V, E)$ tal que $E \subseteq \{\{u, v\} \mid u, v \in V\}$, es decir, E es un conjunto de pares no ordenados de elementos de V. Los elementos de V son los vértices (nodos o puntos) del grafo G, los elementos de E son las aristas (o lineas). Comúnmente se dibuja representando los vértices como puntos y la unión de dos vértices por medio de una línea llamada arista [6]. Un ejemplo de grafo se observa en la Figura 2.1.

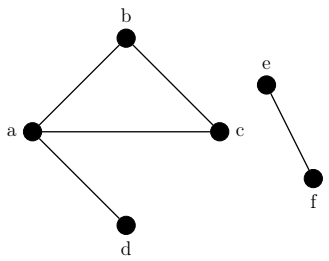


Figura 2.1: Esta figura representa un grafo donde sus nodos son puntos y sus aristas son las líneas que unen los puntos. En este caso, el conjunto de los vértices es $V = \{a, b, c, d, e, f\}$ y el conjunto de las aristas es $E = \{\{a, b\}, \{a, c\}, \{a, d\}, \{b, c\}, \{e, f\}\}$

Dentro de un grafo, se pueden reconocer diversas estructuras o conceptos fundamentales que describen relaciones y propiedades. Un concepto importante es el de nodos ***adyacentes*** o ***vecinos***. Esto ocurre cuando dos vértices x, y de G están unidos por una arista. Otro concepto relevante son los ***subgrafos***, esto sucede cuando, dados dos grafos $G = (V, E)$ y $G' = (V', E')$, si $V' \subseteq V$ y $E' \subseteq E$ entonces G' es ***subgrafo*** de G [6].

Los grafos también se pueden clasificar según sus características y aplicaciones. Un tipo de clasificación son los ***grafos completos***. Estos son aquellos en los que todos los vértices del grafo son adyacentes entre sí. Es decir, si un grafo G tiene n vértices, contendrá $\frac{n(n-1)}{2}$ aristas. Los grafos completos son denotados como K_n, donde n es el número de vértices. Además, si un subgrafo es completo, se le llama ***clique***.

Otro tipo de grafos son los ***grafos aleatorios***. Estos grafos se construyen añadiendo aristas entre los vértices según una probabilidad específica, modelando redes en situaciones donde la conectividad es incierta, como en redes sociales o biológicas.

También se encuentran los ***grafos de intervalo unitarios***. Estos grafos pueden ser representados mediante intervalos en la recta real, donde cada vértice corresponde a un intervalo, de tamaño una unidad, y existe una arista entre dos vértices si sus intervalos se superponen. Un ejemplo de estos grafos se puede apreciar en la Figura 2.2.

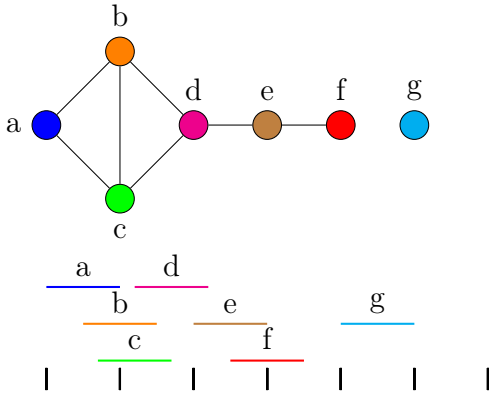


Figura 2.2: Esta figura muestra un grafo de intervalo unitario junto con su representación en intervalos. En la representación en intervalos, se puede observar que las rectas que representan los nodos y sus vecinos se superponen. En el caso del nodo g, se observa que no tiene vecinos, por lo que en el intervalo su recta no se superpone con ninguna otra recta.

Para esta tesis, el tipo de grafo a utilizar son los grafos con signos. Un ***grafo con signos*** se define como un grafo $G = (V, E)$ en el que a cada arista $e \in E$ se le asigna un signo, positivo o negativo. Formalmente, el conjunto de aristas E se divide en dos subconjuntos E^+ y E^-, donde $E = E^+ \cup E^-$ y $E^+ \cap E^- = \emptyset$. Aquí, E^+ representa el conjunto las aristas con signos positivos y E^- el conjunto de las aristas con signos negativos. Un ejemplo de un grafo con signos se observa en la Figura 2.3.

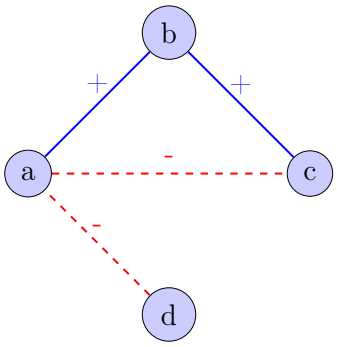


Figura 2.3: Esta figura presenta un grafo con signo. Las líneas continuas representan aristas positivas y las líneas discontinuas representa aristas negativas.

Dentro de los grafos con signos, también se tiene diversas estructuras. Una de ellas es ***subgrafo positivo***. Un subgrafo positivo existe cuando, dado un grafo positivo $G = (V, E)$, existe un grafo $G' = (V', E^+)$, tal que $V' = V$ y $E^+ \subseteq E$, entonces G' es subgrafo positivo del grafo G. Si el subgrafo positivo es completo, entonces es un ***clique de aristas positivas***.

Es necesario definir el siguiente concepto. Un ***orden*** π de un grafo G, es una inyección del conjunto de vértices del grafo G, tal que, $\pi : V \to \{1, 2, 3, \ldots, n\}$, con $n = |V|$ el número de vértices del grafo del G.

2.2. Problema

El problema que se busca resolver es de naturaleza social y se centra en un conjunto de personas que se deben sentar en una mesa. Se sabe de antemano que algunas personas, al sentarse cerca, crean un ambiente agradable, mientras que otras deben mantenerse alejadas debido a sus malas relaciones. El objetivo principal es asegurar que cada individuo esté más cerca de las personas con las que tiene una buena relación que de aquellas con las que tiene una mala relación. Si

una persona se encuentra más cerca de alguien con quien tiene una mala relación que de alguien con quien tiene una buena relación, se genera un conflicto.

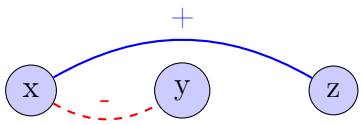
El problema se puede representar mediante un grafo con signos, donde el conjunto de nodos corresponde al conjunto de personas, y las aristas representan las relaciones entre ellas. Las aristas se etiquetan con signos positivos (+) si las personas tienen una buena relación y con signos negativos (-) si tienen una mala relación. La situación en la que se genera un conflicto se denomina *error en un vértice*. Formalmente, el error en un vértice se define de la siguiente forma:

Definición 2.1. *(Error en un vértice) Sea $G = (V, E)$ un grafo con signo y π un orden del conjunto de vértices de G. Un* error en el vértice *x ocurre si existe una tripleta de vértices que involucra a x y se presenta en alguna de las siguientes formas:*

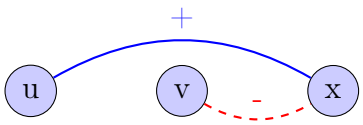
- Error en x por la derecha *ocurre si existen $y, z \in V$ tales que $\pi(x) < \pi(y) < \pi(z)$, con $\{x, y\} \in E^-$ y $\{x, z\} \in E^+$.*
- Error en x por la izquierda *ocurre si existen $u, v \in V$ tales que $\pi(u) < \pi(v) < \pi(x)$, con $\{x, v\} \in E^-$ y $\{x, u\} \in E^+$.*

Se denota por ϵ_x^π el número total de errores en x dados por el orden π.

La Figura 2.4(a) muestra un error en el vértice x cuando se produce por la derecha. Mientras que la Figura 2.4(b) muestra un error en el vértice x cuando se produce por la izquierda.



(a) Error en x por la derecha



(b) Error en x por la izquierda

Figura 2.4: Esta figura presenta ejemplos de errores en los vértices (a) presenta el error en el vértice x por la derecha, (b) presenta el error en el vértice x por la izquierda. Las líneas continuas representan aristas positivas y las líneas discontinuas representan aristas negativas.

Con el error en un vértice definido y siguiendo el punto de vista social, se pueden considerar las siguientes perspectivas. Una vez que las personas están sentadas, si se considera que se sientan en línea recta, se puede observar cuántos ***errores totales*** hay. Lo esperado es que no haya errores, pero puede ocurrir que los haya.

En el caso de que haya errores, se podrían explorar formas alternativas de contarlos. Por ejemplo, en lugar de concentrarse en el total de errores, se puede analizar cuál es el ***máximo número de errores correspondiente a un vértice***. En este caso, si en lugar de enfocarse en los errores totales, se establece un límite a cuántos errores puede soportar un vértice, sería necesario determinar el máximo número de errores por vértice.

Otro enfoque sería contar el ***número de vértices con al menos un error***. Esto permite visualizar si los errores totales se concentran en un conjunto de vértices o están más distribuidos.

Lo anterior, plantea tres formas de medir la calidad de un orden, por lo cual se define lo siguiente:

Definición 2.2. *Dado un grafo con signos G y un orden π del conjunto de vértices de G, se presentan las siguientes métricas para medir la calidad de un orden:*

1. *El* ***número total de errores*** *de π es:*

$$ET(\pi) = \sum_{x \in V} \epsilon_x^\pi.$$

2. *El* ***máximo número de errores correspondiente a un vértice*** *es:*

$$EM(\pi) = \max\{\epsilon_x^\pi : x \in V\}.$$

3. *El* ***número de vértices con al menos un error*** *es:*

$$EV(\pi) = \sum_{\substack{x \in V \\ \epsilon_x^\pi > 0}} 1.$$

Con las métricas definidas, es fácil pensar en cómo usar estas tres métricas.

El trabajo de esta tesis consiste en tomar estas tres métricas y buscar el orden en el que los valores de las métricas sean mínimos.

Volviendo al punto de vista social, al sentar a las personas se busca que las situaciones conflictivas sean mínimas, es decir, se busca que los errores totales sean mínimos.

A su vez, también se podrían aplicar las métricas *EM* (máximo número de errores por vértice) y *EV* (número de vértices con al menos un error). Por ejemplo, si es necesario considerar a todas las personas, se podría aplicar la métrica *EM*. Esto sería buscar ser equitativo en la distribución de los errores. Al haber equidad, las personas podrían considerar que, pese a estar cerca de personas con mala relación, buscarían no generar conflictos.

En caso contrario, si se puede dejar a algunas personas fuera de la mesa, se podría considerar la métrica *EV*. Al minimizarla, se buscaría la menor cantidad de personas que deberían ser excluidas del conjunto para poder sentar al resto de las personas sin generar errores.

Así, el problema de estudio de esta tesis es el siguiente:

Problema: *Dado un grafo con signos G, y alguna de las tres métricas definidas en el párrafo anterior, encontrar un orden π, que minimice la métrica deseada.*

Dado que se consideran tres métricas, la resolución del problema requiere abordarlas de manera independiente. Esto implica que, en total, se deben analizar y resolver tres problemas los cuales se pueden abordar de forma similar.

Encontrar un orden que minimice la primera métrica es un problema ya estudiado, definido anteriormente como el *Problema MinSA* en [25]. La motivación para estudiar el problema MinSA, que es un problema de optimización, surge de la búsqueda de otro tipo de solución al problema de decisión ***Sitting Arrangement (SA)***. El problema SA y MinSA se definirán en el siguiente capítulo. Además, en el mismo trabajo se presentó la heurística BVNS, el cual resolvió el problema MinSA con muy buenos resultados.

En esta tesis, se busca mejorar los resultados obtenidos por la heurística BVNS implementando algoritmos genéticos combinados con el algoritmo t-SNE. Además, se busca implementar estos algoritmos, tanto los genéticos como el BVNS, en los problemas relacionados con las dos métricas restantes.

En los siguientes capítulos, se contextualizará más sobre el problema SA y MinSA. También se describirá en más detalle la heurística BVNS, los algoritmos genéticos y t-SNE. Se buscarán los mejores parámetros para el algoritmo t-SNE y el algoritmo genético. Finalmente, se concluirá con la comparación del rendimiento de los algoritmos.

Estado del arte

En este capítulo se presentará una revisión de los estudios y desarrollos relacionados con el *problema SA (Sitting Arrangement)* y el problema de optimización *MinSA*, que surge a partir del *problema SA*.

El problema *Sitting Arrangement (SA)* es un problema de decisión que ha sido ampliamente estudiado en el contexto de la teoría de grafos, principalmente en los grafos que representan relaciones binarias. Este problema se define como:

Definición 3.1. *Dado un grafo con signo $G = (V, E)$ y un espacio métrico $(\mathcal{M}, d)$, ¿existe un embedding $D : V \to \mathcal{M}$ tal que, para cada par de aristas $\{x, y\}, \{x, w\}$ con $\{x, y\} \in E^+$ y $\{x, w\} \in E^-$, el embedding satisface $d(D(x), D(y)) < d(D(x), D(w))$? Donde d es la distancia en el espacio métrico $\mathcal{M}$.*

Es claro que el problema SA es un problema de decisión, dado que las respuestas posibles son: *Sí existe un embedding* y *No existe un embedding*. Este problema fue introducido inicialmente por Kermarrec y Thraves en [16]. Los autores, en lugar de hablar de embedding, presentaron la noción de *dibujo válido*, ambas definiciones son equivalentes. Además, los autores demostraron que, en un grafo con signo completo y el espacio métrico de la línea euclidiana, se puede determinar en tiempo polinomial de $O(n + m)$ si existe un embedding o no. En caso de que exista un embedding, propusieron un algoritmo que lo encuentra en tiempo polinomial.

Posteriormente, Cygan en [5] demostró que el problema SA es **NP-completo** cuando se considera el caso general, en el que los grafos con signos no son exclusivamente completos. También presentó una caracterización alternativa de los grafos completos con signo y con embedding en la recta, demostrando que un

grafo completo con signo posee un embedding si y solo si su subgrafo positivo es un grafo de **intervalo unitario**.

Pardo, Soto y Thraves, en [25], motivados por el *problema SA*, buscaron una solución para la versión de optimización del *problema SA* en la recta euclidiana. Así, se planteó el *problema MinSA*, el cual consiste en minimizar el número de errores producidos en el embedding. Un *error* es un par de aristas incidentes x, y, x, w tales que $x, y \in E^+$, $x, w \in E^-$ y $|D(x), D(y)| \geq |D(x), D(w)|$. Lo anterior es equivalente a la definición 2.1.

Pardo, Soto y Thraves demostraron que, en grafos completos, sus mínimos locales coinciden con los del problema *Quadratic Assignment (QA)* en un caso particular. Además, en el mismo trabajo, propusieron dos heurísticas basadas en un enfoque voraz (greedy), una de ellas siguiendo la metodología GRASP. Los experimentos mostraron que GREEDY encuentra la solución óptima cuando el grafo tiene cero errores, mientras que GRASP es ligeramente mejor en grafos sin estructura especial, aunque sin diferencias estadísticas significativas entre ambas. Además, las heurísticas diseñadas para el problema QA también ofrecen resultados comparables bajo la función objetivo de MinSA, lo que sugiere una fuerte conexión entre ambos problemas.

Más recientemente, Pardo, García-Sánchez, Sevaux, y Duarte en [27], se presentaron la heurística Basic Variable Neighborhood Search (BVNS) para resolver el problema MinSA, el cual es una variante de la metaheurística Variable Neighborhood Search (VNS). En el trabajo, primero se definió un pre-proceso constructivo. Este proceso consiste en un nuevo esquema de construcción basado en la identificación de cliques dentro del grafo de entrada, considerando únicamente las aristas positivas. Este enfoque plantea que los vértices de un clique no generan errores entre ellos, por lo que son adecuados para ubicarse juntos en un orden. Luego, el orden generado se utiliza como punto de partida para el algoritmo BVNS.

Adicionalmente, se comparó BVNS con los mejores enfoques previos en la literatura. Los resultados de esta comparación indicaron que BVNS superaba a los métodos anteriores, lo cual fue aún más respaldado mediante pruebas estadísticas no paramétricas, consolidándolo como el nuevo algoritmo de última generación para el problema MinSA.

Parte de esta tesis consiste en tomar el algoritmo BVNS y compararlo con la implementación del Algoritmo Genético, buscando obtener mejores resultados para el problema MinSA. El objetivo es evaluar el desempeño de cada algoritmo por separado, analizando su capacidad para minimizar el número de errores en un orden. Esto implica diseñar experimentos detallados que permitan comparar el desempeño de ambos algoritmos en diferentes escenarios, incluyendo grafos con diversas características y tamaños.

Técnicas de optimización a implemetar

En este capítulo, se presentarán con mayor detalle las dos técnicas de optimización a comparar: Basic variable Neighborhood Search (BVNS) y los Algoritmos Genéticos (AG). Ambos métodos son ampliamente utilizados para resolver problemas combinatorios complejos, pero difieren significativamente en su enfoque y estrategia de búsqueda.

BVNS se basa en la exploración sistemática de diferentes vecindarios, que son conjuntos de soluciones alcanzables mediante movimientos o transformaciones específicas a partir de una solución dada. Estos movimientos pueden considerarse como las operaciones necesarias para pasar de una solución a otra. Los vecindarios pueden variar en tamaño y estructura, lo que permite alternar entre búsquedas locales intensivas y perturbaciones controladas para escapar de óptimos locales. Por ejemplo, en un problema de optimización combinatoria, un vecindario podría incluir todas las soluciones que difieren de la solución actual en un número limitado de variables, como intercambiar un elemento por otro o cambiar una componente de la solución. Esta técnica es efectiva en problemas donde la estructura de la solución permite definir vecindarios eficientes, facilitando la mejora iterativa de una solución inicial.

Por otro lado, los Algoritmos Genéticos (AGs) se basan en principios de evolución biológica, comenzando con una población inicial que evoluciona a lo largo de las generaciones mediante procesos de selección, cruce y mutación. Este enfoque promueve la diversidad y la exploración global del espacio de búsqueda, lo que

resulta particularmente ventajoso en espacios no convexos o multimodales. En estos espacios, la función objetivo puede presentar múltiples óptimos locales, y los AGs son efectivos porque exploran simultáneamente diferentes regiones del espacio de soluciones, aumentando las posibilidades de encontrar el óptimo global. A diferencia del BVNS, que mejora una solución iterativamente, los AGs trabajan con múltiples soluciones simultáneamente, proporcionando una cobertura más amplia del espacio de soluciones posibles y reduciendo significativamente la probabilidad de quedar atrapados en un óptimo local.

En las siguientes secciones, se profundizará en cada uno de estos algoritmos, explorando sus principios fundamentales, ventajas y aplicaciones en el contexto del problema MinSA.

4.1. BVNS

Basic Variable Neighborhood Search (BVNS) es una variante de la metaheurística Variable Neighborhood Search (VNS), que explora diferentes vecindades para escapar de óptimos locales. VNS fue introducida por Mladenović y Hansen en 1997 [23], y ha sido ampliamente utilizada en la optimización de problemas difíciles. Con el tiempo, VNS ha evolucionado en múltiples variantes [12], consolidándose como un marco general para el desarrollo de heurísticas y una herramienta poderosa en optimización.

Otras variaciones populares de esta metaheurística incluyen Reduced Variable Neighborhood Search (RVNS), Variable Neighborhood Descent (VND), y General Variable Neighborhood Search (GVNS) [12]. Además, existen estrategias como Variable Framework Search (VFS) [24], para problemas máx/mín–mín/máx, Multi-Objective Variable Neighborhood Search (MO-VNS) [8], para optimización multiobjetivo, y Parallel Variable Neighborhood Search (PVNS) [9]. Esta última aprovecha específicamente las tecnologías actuales de computación paralela y distribuida, permitiendo utilizar múltiples procesadores o núcleos para ejecutar búsquedas en paralelo, lo que mejora significativamente la eficiencia y velocidad de la búsqueda. BVNS para el problema MinSA combina exploración determinista y aleatoria de vecindades para mejorar soluciones iterativamente. Este algoritmo se puede dividir en tres principales pasos:

- Shake: Consiste en aplicar un movimiento aleatorio dentro de una vecindad

sobre la solución actual. Los movimientos posibles incluyen insertar o intercambiar elementos. Aunque el tipo de movimiento (insertar o intercambiar) se fija antes de ejecutar el algoritmo, el proceso de elegir qué vértice realizará el movimiento es aleatorio.

- LocalSearch: Mejora la solución anterior mediante una búsqueda local. Al igual que el paso Shake, los posibles movimientos en LocalSearch incluyen insertar o intercambiar. Por lo tanto, se definen dos variantes de LocalSearch: LS1 y LS2. LS1 utiliza movimientos basados en inserciones, mientras que LS2 utiliza movimientos basados en intercambios.
- NeighborhoodChange: Realiza una comparación entre la solución inicial y la solución obtenida después de los dos pasos anteriores. Se selecciona la solución que mejor optimiza la función objetivo. Luego, se decide si es necesario repetir los pasos anteriores o si se puede finalizar el proceso.

Además, BVNS cuenta con un límite de tiempo y un límite del tamaño máximo de vecindad a explorar, lo que ayuda a controlar la complejidad computacional del proceso.

Algoritmo Constructivo. Para mejorar el funcionamiento del BVNS, una técnica alternativa es utilizar un orden entregado por un algoritmo constructivo como entrada. Este algoritmo constructivo se basa en la identificación de cliques dentro del grafo de entrada, considerando únicamente las aristas positivas. Esto ayuda, ya que este tipo de estructuras no presentan errores al realizar el orden. Así, el orden generado por el algoritmo constructivo es un buen punto de partida para BVNS. La ejecución del algoritmo constructivo consiste en trabajar con el subgrafo positivo del grafo con signos de entrada, lo cual facilita la identificación de los cliques de aristas positivas. Para generar el orden, se toman los vértices del clique de mayor tamaño y se agregan al orden. Luego, se toma el siguiente clique de mayor tamaño que no contenga los vértices ya agregados al orden. Este proceso continúa hasta que no falte ningún vértice por agregar al orden.

A continuación, se presentarán con mayor detalle los pasos Shake y Local Search del algoritmo BVNS, junto con el algoritmo constructivo y su pseudocódigo. Además, se proporcionará una explicación más detallada del funcionamiento del BVNS, acompañada de un pseudocódigo que ilustre su implementación.

4.1.1. Shake

Shake es un paso clave para diversificar la búsqueda y evitar quedar atrapado en óptimos locales. Cuando el algoritmo llega a este paso, su función principal es generar una nueva solución dentro del vecindario actual para continuar la exploración del espacio de soluciones. Este proceso se centra en la generación aleatoria de una nueva solución a partir de la solución actual.

Durante el paso Shake, se aplica un número determinado de movimientos a la solución actual. Esto implica realizar modificaciones aleatorias en la solución actual para obtener una nueva solución. Esta estrategia permite al algoritmo explorar diferentes áreas del espacio de soluciones, lo que es fundamental para evitar que el BVNS se estanque en un óptimo local.

Se proponen dos procedimientos Shake diferentes, basados en operadores de movimiento clásicos utilizados en problemas de diseño lineal: inserción e intercambio. En el movimiento de inserción, se elimina un vértice de su posición actual i y se inserta en una nueva posición j, generando así una nueva solución. Por otro lado, el movimiento de intercambio consiste en eliminar el vértice en la posición i y colocarlo en la posición j, mientras que simultáneamente se elimina el vértice actualmente ubicado en la posición j y se coloca en la posición i. Ambos movimientos producen nuevas soluciones que permiten al algoritmo explorar diferentes configuraciones.

Además, cada uno de estos procedimientos Shake se utiliza dependiendo de la estrategia de LocalSearch empleada dentro del BVNS. Es decir, si LocalSearch emplea la variante LS1, que utiliza inserción, entonces Shake usará un procedimiento basado en intercambio. En caso contrario, si LocalSearch utiliza la variante LS2,que utiliza el intercambio, Shake realizará el procedimiento de inserción. Esta combinación de procedimientos ayuda al algoritmo a evitar quedar atrapado en soluciones que no son las mejores y a explorar diferentes partes del espacio de soluciones. De esta manera, mejora su capacidad para encontrar las soluciones más óptimas.

4.1.2. LocalSearch

LocalSearch, en el contexto del problema MinSA, es un proceso de mejora iterativa de una solución existente mediante la exploración de soluciones vecinas. Se utilizan

movimientos específicos, como inserción e intercambio, para modificar la solución actual y evaluar si la nueva solución es mejor.En el trabajo realizado por Pardo, García-Sánchez, Sevaux y Duarte, propusieron dos métodos de búsqueda local: LS1 y LS2 [27].

Ambas estrategias se basan en uno de los dos movimientos mencionados anteriormente, inserción e intercambio, para luego ver si el movimiento realizado produce un cambio favorable o no en el orden. LS1 utiliza el movimiento de inserción para encontrar un óptimo local, mientras que LS2 emplea el movimiento de intercambio bajo la misma estrategia. La implementación directa de estos métodos requiere que, después de cualquier movimiento, se actualice el número de errores que cada vértice involucrado en este movimiento. Por lo tanto, en el peor de los casos, cuando es un grafo completo, la complejidad computacional para evaluar exactamente cada movimiento es $O(\mid V \mid \mid E \mid)$. Sin embargo, es posible identificar qué vértices podrían verse afectados por un movimiento específico en el orden, así que en los casos que no fuera un grafo completo la complejidad computacional bajaría. La función objetivo del problema MinSA se calcula sumando los errores en cada vértice. Un error ocurre cuando un vecino negativo se coloca más cerca que un vecino positivo en el orden. Esto depende únicamente de cómo están colocados los vértices adyacentes entre sí.

Con estas definiciones claras, se puede observar que, para movimientos de inserción, las únicas posiciones relativas que podrían cambiar son aquellas relacionadas con el vértice que está siendo movido y sus vértices adyacentes. Esto permite optimizar el proceso de actualización al enfocarse solo en los vértices directamente afectados, en lugar de re-calcular todos los errores en la solución.

El objetivo general es encontrar un óptimo local, es decir, una solución que sea mejor que todas sus vecinas. La búsqueda local es un componente clave en algoritmos más complejos, como BVNS, para escapar de óptimos locales y encontrar soluciones de alta calidad. La combinación de LS1 y LS2 permite diversificar la búsqueda y mejorar la efectividad del proceso de optimización.

4.1.3. Algoritmo Constructivo

El algoritmo constructivo recibe como entrada un grafo con signo G y un tiempo límite permitido para calcular los cliques positivos. Luego, se crea un conjunto de

todos los cliques con aristas positivas posibles que pertenecen a G, utilizando el algoritmo de Bron-Kerbosch. El tamaño de los cliques en el conjunto varía entre uno y el tamaño máximo del grafo, y están ordenados de mayor a menor tamaño. Si el tiempo de cálculo para crear el conjunto de cliques es inaceptablemente grande, se detiene el método después de un cierto umbral, lo que proporciona una lista de cliques que al menos cumplen un criterio de tamaño mínimo. Posteriormente, con el conjunto de cliques, se crea el orden.

El algoritmo de Bron-Kerbosch [4], basado en una estrategia recursiva con retroceso (backtracking), es particularmente útil en este contexto. Este algoritmo permite identificar todos los cliques presentes en el grafo de manera eficiente, lo que es crucial para generar el conjunto de cliques positivos necesarios para crear el orden. Al utilizar este algoritmo, se evita la búsqueda exclusiva del clique de mayor tamaño, lo que facilita la creación del conjunto de cliques en un tiempo razonable. Además, este enfoque permite obtener una lista completa de cliques, lo que es beneficioso para asegurar que el orden generado sea lo más óptimo posible dentro del tiempo disponible.

Un pseudocódigo del algoritmo Constructivo se presenta en Algoritmo 1. Se requieren como entrada la instancia del grafo G y un valor máximo de tiempo de ejecución, t_{limit}. El algoritmo comienza con una solución vacía, φ. Después se eliminan todas las aristas negativas del grafo utilizando el algoritmo de Bron-Kerbosch (paso 3). Luego, en cada iteración del algoritmo (pasos 5 - 13), el procedimiento selecciona el clique más grande disponible en el paso 6 (getLargestClique). En el paso 7, los vértices dentro de los clique seleccionados se asignan a en el orden. Finalmente, en los pasos 8–12, el método recorre todas las clique aún en C y elimina todos aquellos clique que contienen al menos un vértice ya seleccionado en el orden [27] .

En el trabajo que planteo este algoritmo, [27], se propusieron tres estrategias para asignar el clique más grande en el orden (AssignConsecutiveLabels). El método elegido para realizar este paso fue una estrategia llamada C3. Esta estrategia consiste en insertar un clique entre cualquier par de cliques previamente asignados en la solución parcial. De esta manera, se busca insertar el clique en una ubicación favorable para asegurar que se está construyendo en el mejor orden posible.

Algorithm 1 Construvtive algorithm

```
1: procedure CONSTRUCTIVE(G, t_limit)
2:     φ ← ∅
3:     G+ ← removeNegativeEdges(G)
4:     C ← BUILDCLIQUES(G, t_limit)
5:     while (C ≠ ∅) do
6:         c* ← GETLARGESTCLIQUE(C)
7:         ASSINGCONSECUTIVELABELS(φ, c*)
8:         for each c ∈ C do
9:             if (INTERSECTION(c, c*)) then
10:                REMOVE(c, C)
11:            end if
12:        end for
13:    end while
14:    return φ
15: end procedure
```

4.1.4. BVNS pseudocódigo

Un pseudocódigo del algoritmo BVNS se presenta en Algoritmo 2. Inicialmente se requieren como entrada un orden φ, un limite de vecindario k_{max} y un tiempo limite t_{limit}. Luego se inicializa un valor k en uno, lo que representa el tamaño de vecindad a explorar. Después se ingresa al bucle principal (pasos 5, 6 y 7). En este bucle, primero se toma la solución actual, φ, y se perturba en el procedimiento Shake, aplicando un movimiento aleatorio en el vecindario k. Luego, la solución obtenida se mejora en el procedimiento LocalSearch. Finalmente, se utiliza una implementación del método NeighborhoodChange. En este procedimiento se determina que vecindario es el siguiente a explorar. Si la solución perturbada (φ'') mejora a la solución actual (φ), el método BVNS considera φ'' como la nueva solución actual ($\varphi \leftarrow \varphi''$) y se reinicia el vecindario de búsqueda a uno ($k \leftarrow 1$). En caso contrario, donde la solución perturbada no mejora la solución actual, se mantiene el valor de la solución actual y el tamaño del vecindario aumenta en uno ($k \leftarrow k + 1$). El bucle principal se repite hasta que se explora el vecindario más grande (k_{limit}) sin encontrar una mejora. Luego, si no se ha excedido el tiempo máximo (t_{limt}), el algoritmo realiza una nueva iteración, es decir, se inicia el bucle nuevamente. Si el tiempo es excedido, el algoritmo devuelve el orden encontrado [27].

Algorithm 2 BVNS algorithm

```
1: procedure BVNS(φ, k_max, t_max)
2:     repeat
3:         k ← 1
4:         repeat
5:             φ' ← Shake((φ, k))
6:             φ'' ← LocalSearch((φ'))
7:             NeighborhoodChange((φ', φ'', k))
8:         until k > k_max
9:         t ← CPUTime()
10:    until t ≥ t_max
11:    return φ
12: end procedure
```

4.2. Algoritmos Genéticos

Los Algoritmos Genéticos (AGs), inspirados en la selección natural y la genética, fueron desarrollados por John Holland y sus colegas en la Universidad de Michigan con el objetivo de modelar procesos adaptativos y aplicar estos principios a sistemas computacionales. El objetivo era diseñar algoritmos que imitaran la evolución biológica, donde una población de individuos evoluciona a lo largo de generaciones, para resolver problemas complejos de optimización y búsqueda. A través de la creación de nuevas soluciones combinando partes de las mejores soluciones anteriores, se permite una mejora progresiva en cada generación. A pesar de su naturaleza aleatoria, los AGs aprovechan la información histórica para guiar la búsqueda de manera eficiente, lo que los diferencia de una simple exploración aleatoria. Un concepto clave en este enfoque es la robustez, que busca equilibrar la eficiencia y la eficacia en diversos entornos. La capacidad de adaptación de los AGs ha impulsado su aplicación en la optimización de funciones, el control de sistemas y otros campos, con las bases teóricas establecidas en *Adaptation in Natural and Artificial Systems (1975)* de Holland [11].

Desde una perspectiva de optimización, un AGs es un método de búsqueda probabilístico que opera sobre una población de soluciones. En algoritmos elitistas, donde siempre se conserva la mejor solución encontrada, se ha demostrado que la convergencia hacia la solución óptima aumenta con cada iteración. Este proceso imita la evolución natural, donde los individuos (soluciones) se reproducen mediante operadores genéticos y los más aptos (eficientes) son seleccionados. Para

aplicar esta idea a problemas de optimización, es necesario establecer equivalencias clave, representar las soluciones de manera análoga a los cromosomas, definir la adecuación de cada solución como una medida de su rendimiento (normalmente el valor a optimizar), y establecer operadores genéticos que simulen la recombinación y mutación para modificar y mejorar las soluciones a lo largo del tiempo

Las partes clave de un algoritmo genético incluyen la inicialización de la población, donde se crean las primeras soluciones de manera aleatoria; la evaluación de la aptitud, que determina la calidad de cada solución; la selección, que elige las soluciones más aptas para reproducirse; los operadores genéticos como el cruce (recombinación) y la mutación, que generan nuevas soluciones; y la repetición del ciclo hasta alcanzar un criterio de parada, como un límite de generaciones, una mejora insuficiente en la aptitud o un límite de tiempo. Estos componentes trabajan juntos para explorar el espacio de soluciones de manera eficiente y converger hacia soluciones óptimas [1].

De lo anterior se destacan los siguientes operadores genéticos:

- *Selección*: puede realizarse mediante varios métodos, como la selección por torneos, donde se enfrentan soluciones en parejas y la más apta avanza, o la selección por ruleta, donde las soluciones más aptas tienen una probabilidad mayor de ser seleccionadas. Estos métodos de selección aseguran que las soluciones más prometedoras tengan más oportunidades de reproducirse y contribuir a las generaciones futuras. Además, técnicas como la selección elitista garantizan que las mejores soluciones siempre se conserven en la población, lo que ayuda a mantener la calidad de las soluciones a lo largo del proceso de optimización.
- *Cruce* (Recombinación): Este operador combina partes de dos soluciones parentales para crear nuevas soluciones, lo que permite la mezcla de características prometedoras. Por ejemplo, si estamos optimizando una función que depende de dos variables, las soluciones podrían representarse como pares de números. El cruce podría intercambiar los valores de una variable entre dos soluciones, creando dos nuevas soluciones que combinan aspectos de ambas. Este proceso permite explorar nuevas combinaciones de características que podrían mejorar la calidad de las soluciones.
- *Mutación*: Introduce variaciones aleatorias en las soluciones existentes,

evitando que el algoritmo quede atrapado en óptimos locales. Por ejemplo, si tenemos una solución representada por un vector de números, la mutación podría cambiar aleatoriamente uno de esos números. La mutación es crucial para mantener la diversidad en la población y asegurar que el algoritmo explore diferentes áreas del espacio de soluciones.

AGs en el problema MinSA: Para implementar un algoritmo genético en el problema de estudio, se diseñaron distintas estrategias para mejorar la población inicial y el operador de mutación. Para la población inicial, se consideró utilizar la técnica de reducción de dimensionalidad t-Distributed Stochastic Neighbor Embedding (t-SNE). El uso del t-SNE se debe a que, con la ayuda de esta técnica, se puede reducir la dimensionalidad del grafo a una dimensión, lo cual se asemeja a asignar un orden a los vértices. De esta manera, se obtiene una población inicial que brinda un buen punto de partida para el algoritmo genético.

También se creó una función de mutación llamada SwapAndInsert, la cual está inspirada en la estrategia de insertar e intercambiar nodos en un orden que se utiliza en los pasos Shake y LocalSearch del algoritmo BVNS. El uso de estos movimientos se debe a que brindan nuevas soluciones de forma aleatoria, lo que ayuda a evitar caer en mínimos locales.

A continuación se presentarán más en detalle el funcionamiento del t-SNE y su implementación. Luego se explicará la función de mutación creada. Finalmente se presentará en detalle por medio de un pseudocódigo el Algoritmo Genético.

4.2.1. t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE (t-Distributed Stochastic Neighbor Embedding) es una técnica estadística de reducción de dimensionalidad propuesta por Laurens van der Maaten y Geoffrey Hinton en 2008 [30]. Es es una variación de Stochastic Neighbor Embedding (SNE), introducido por Hinton y Roweis en 2002, optimizando su proceso de cálculo y mejorando significativamente la calidad de las visualizaciones generadas. El objetivo del t-SNE es proyectar datos de alta dimensión en un espacio de menor dimensión, generalmente bidimensional o tridimensional, preservando las relaciones locales entre los puntos de datos. Esto facilita la visualización de estructuras complejas dentro de los datos, abordando uno de los principales

desafíos en el análisis de datos de alta dimensión: encontrar representaciones visuales que reflejen con fidelidad su estructura subyacente.

4.2.1.1. Funcionamiento

El algoritmo t-SNE transforma un conjunto de datos en alta dimensión $X = \{x_1, x_2, ..., x_n\}$ en una representación de menor dimensión $y = \{y_1, y_2, ..., y_n\}$, manteniendo la mayor cantidad de información posible. Para lograrlo, el algoritmo busca que las relaciones de similitud entre pares de puntos en el espacio original (p_{ij}) se mantengan lo más fielmente posible en el espacio reducido (q_{ij}).

Las similitudes por pares p_{ij} se definen con distribuciones gaussianas:

- $p_{j|i}$: Es la probabilidad condicional de que el punto x_j sea elegido como vecino de x_i, asumiendo que la distribución de vecinos sigue una distribución gaussiana centrada en x_i. Se calcula como:

$$p_{j|i} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|x_i - x_k\|^2 / 2\sigma_i^2\right)} \tag{4.1}$$

  donde σ_i es el parámetro de dispersión (desviación estándar) para el punto x_i, ajustado para alcanzar la perplejidad deseada.

- p_{ij}: Es la probabilidad simétrica de similitud entre x_i y x_j, definida como el promedio de $p_{j|i}$ y $p_{i|j}$:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n} \tag{4.2}$$

  donde n es el número de puntos en el conjunto de datos.

Es decir, $p_{j|i}$ representa la probabilidad condicional asimétrica de elegir x_j como vecino de x_i, mientras que p_{ij} es una versión simétrica utilizada en la comparación con el espacio de baja dimensión.

Para garantizar que la distribución $p_{j|i}$ capture la densidad local correctamente, se ajusta σ_i de manera que la **entropía de Shannon**:

$$H(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i} \tag{4.3}$$

coincida con un valor deseado determinado por la **perplejidad**:

$$\text{Perp}(P_i) = 2^{H(P_i)} \tag{4.4}$$

Este ajuste se realiza mediante búsqueda por bisección para asegurar que la cantidad de vecinos efectivos de cada punto sea aproximadamente igual a la perplejidad especificada.

Para representar los puntos en el espacio de menor dimensión y_i, se define una distribución basada en una **distribución t de Student** con 1 grado de libertad:

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq m}(1 + \|y_k - y_m\|^2)^{-1}} \tag{4.5}$$

Esta distribución minimiza la influencia de puntos lejanos, lo que ayuda a manejar mejor la congestión en espacios de baja dimensión.

Para encontrar la mejor representación en el espacio reducido, t-SNE minimiza la divergencia de Kullback-Leibler entre las distribuciones P y Q:

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{4.6}$$

El descenso de gradiente se utiliza para minimizar esta función de costo. La actualización de cada y_i se basa en el gradiente:

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + \|y_i - y_j\|^2)^{-1} \tag{4.7}$$

Finalmente, la actualización de los puntos en el espacio reducido sigue la regla:

$$y_i^{(t)} = y_i^{(t-1)} + \eta \frac{\delta C}{\delta y_i} + \alpha(y_i^{(t-1)} - y_i^{(t-2)}) \tag{4.8}$$

donde η es la tasa de aprendizaje y α un término de momento que ayuda a estabilizar la optimización.

Este proceso permite obtener una representación de los datos en baja dimensión donde las relaciones de similitud entre los puntos se preservan tanto como sea posible [30].

4.2.1.2. Ventajas y aplicaciones

Una de las características más relevantes de t-SNE es su capacidad para revelar estructuras en múltiples escalas, lo que lo hace especialmente útil en datos distribuidos en múltiples subconjuntos interrelacionados, como imágenes de objetos pertenecientes a distintas clases o datos de secuencias temporales complejas. Además, para el análisis de grandes volúmenes de datos, t-SNE puede emplear recorridos aleatorios en grafos de vecindad, permitiendo que la representación visual refleje la estructura implícita del conjunto de datos completo [31].

4.2.1.3. Pseudocódigo

Un pseudocodigo del Algoritmo T-SNE, de como es implementado, se observa en el Algoritmo 3. En este pseudocódigo, primero se ingresa un conjunto de datos de entrada y parámetros de costo y optimización. Luego, se calcula las afinidades por pares entre los puntos de datos, se establece una matriz de afinidad inicial, y se muestrea una solución inicial de baja dimensionalidad. Después, se itera T veces para calcular las afinidades de baja dimensionalidad, el gradiente y actualiza las posiciones de los puntos en el espacio de baja dimensión. Finalmente, se proporciona una representación de datos en baja dimensión como salida.

Algorithm 3 Simple version of t-Distributed Stochastic Neighbor Embedding

Input: Data set $\mathcal{X} = \{x_1, x_2, ..., x_n\}$,
Cost function parameters: perplexity $Perp$,
Optimization parameters: number of iterations T, learning rate η, momentum $\alpha(t)$.
Output: Low-dimensional data representation $\mathcal{Y}^{(T)} = \{y_1, y_2, ..., y_n\}$.

begin
    Compute pairwise affinities $p_{j|i}$ with perplexity $Perp$ (using Equation 4.1)
    Set $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$
    Sample initial solution $\mathcal{Y}^{(0)} = \{y_1, y_2, ..., y_n\}$ from $\mathcal{N}(0, 10^{-4}I)$
for $t = 1$ to T **do**
        Compute low-dimensional affinities q_{ij} (using Equation 4.5)
        Compute gradient $\frac{\delta C}{\delta Y}$ (using Equation 4.7)
        Update positions:
            $\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta Y} + \alpha(t)(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)})$
end for
end

4.2.2. Función de mutación: SwapAndInsert

La función *SwapAndInsert* aplica mutaciones a un individuo en un algoritmo genético mediante dos operadores: intercambio e inserción. Con una probabilidad del 50 %, selecciona dos posiciones aleatorias y permuta sus valores, introduciendo pequeñas variaciones en el orden. Además, con una probabilidad del 50 %, extrae un elemento de una posición aleatoria y lo inserta en otra, modificando la estructura de la permutación sin perder su validez. Estas mutaciones permiten explorar nuevas soluciones dentro del espacio de búsqueda, facilitando la optimización en los problemas de estudio.

4.2.3. AGs pseudocódigo

Un Algoritmo Genético funciona de la siguiente manera: primero, se ingresa un conjunto inicial de soluciones, llamado población, que contiene n individuos. Esta población puede ser creada aleatoriamente o puede definirse de antemano. Luego, se evalúa la calidad (aptitud) de cada individuo en la población.

A continuación, se utiliza el método de selección por torneos para elegir dos cromosomas, que serán los padres para generar descendencia. Este método consiste en seleccionar una cantidad determinada de individuos aleatorios de la población y compararlos según su valor de aptitud para seleccionar a los que serán los padres de la siguiente generación.

Después, se realiza un cruce de los padres en un punto específico, con una cierta probabilidad (p_c), para crear un nuevo hijo. Luego, se modifica aleatoriamente el hijo de manera uniforme, con una probabilidad de mutación (p_m), generando un descendiente aún más nuevo. Este último descendiente se añade a la siguiente generación de la población.

Este proceso de selección, cruce y mutación se repite varias veces en la población actual hasta que la nueva población esté completa.

Un pseudocodigo de lo anterior se observa en Algoritmo 4.

Algorithm 4 Algoritmo Genético

Input: Tamaño de población N, número de generaciones G, probabilidad de cruce p_c, probabilidad de mutación p_m
Output: Mejor solución encontrada
Inicializar la población P aleatoriamente
for cada generación $g = 1$ hasta G **do**
    Evaluar la función de aptitud de cada individuo en P
    Seleccionar padres mediante un método de selección (ej. torneo, ruleta)
    Aplicar cruce con probabilidad p_c para generar descendencia
    Aplicar mutación con probabilidad p_m sobre la descendencia
    Evaluar la descendencia y reemplazar individuos en P si es necesario
end for
return Mejor individuo de P

Experimentos computacionales

En este capítulo, se presentan los resultados obtenidos de los experimentos realizados para evaluar empíricamente el desempeño de los algoritmos propuestos en este trabajo, en relación con los problemas abordados. Inicialmente, se busca determinar los mejores parámetros para el algoritmo genético y comparar su rendimiento con la heurística BVNS en el problema de optimizar el número total de errores. En paralelo, se presenta un análisis que muestra cómo funcionan el algoritmo genético y BVNS cuando se cambia la función objetivo. Esto implica cambiar el enfoque de optimización, pasando de minimizar el número total de errores a dos nuevas opciones: optimizar el máximo número de errores correspondiente a un vértice, o bien, optimizar el número de vértices que tienen al menos un error.

Para realizar las pruebas, se han seleccionado tres conjuntos de experimentación y tres conjuntos de evaluación. Estas instancias se describen en la Sección 5.1. Luego, en la Sección 5.2, se presentan los resultados de los experimentos realizados para ajustar los parámetros de la configuración final.

Los algoritmos han sido implementados en Python 3 y fueron ejecutados en un entorno con un procesador Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, con 16 GB de RAM y un sistema operativo Windows 10 de 64 bits.

5.1. Conjuntos de datos

Para realizar las experimentaciones, se han considerado dos conjuntos: un conjunto de experimentación y un conjunto de evaluación. El conjunto de experimentación

se utiliza para ajustar los parámetros del algoritmo genético, mientras que el conjunto de evaluación se emplea para comparar el rendimiento del algoritmo genético con la heurística BVNS. Se utilizo instancias de conjuntos con las mismas propiedades que los conjuntos utilizados por Pardo, Soto y Thraves [25]. Cabe señalar que unos de los conjuntos tiene como propiedad ser grafos de intervalo unitarios. Esto se debe a que, Kermarrec y Thraves [16], probaron que la estructura de estos grafos permite conocer el valor óptimo por construcción. En particular, cada instancia en este conjunto se puede organizar de tal manera que el número total de errores sea cero.

5.1.1. Conjuntos de experimentación

- **Conjunto completo:** Este conjunto está compuesto por 36 grafos completos. El tamaño de los grafos corresponden a 100 vértices. Para cada grafo, se consideran diferentes porcentajes de aristas negativas 20 %, 50 % y 80 %, generándose 12 grafos para cada porcentaje.

  Primero se generó un grafo completo para el tamaño especificado. En este caso, todas las aristas iniciales fueron asignadas con el valor +1, asegurando que el grafo fuera completamente conectado con aristas positivas. Luego se seleccionaron aristas del grafo según el porcentaje especificado (20 %, 50 % o 80 %). Estas aristas seleccionadas aleatoriamente, se les asignó el valor −1, respetando la cantidad determinada.

- **Conjunto aleatorio:** Este conjunto está compuesto por 36 grafos aleatorios. El tamaño de los grafos corresponden a 100 vértices. Para cada grafo, se consideran una probabilidad fija de arista existente de 50 % y diferentes porcentajes de aristas negativas 20 %, 50 % y 80 %. Para cada combinación de parámetros, se generan 12 grafos.

  Para generar esta conjunto, primero se generó un grafo aleatorio para el tamaño especificado, utilizando una probabilidad fija ($p = 0{,}5$) para la existencia de aristas. Esto asegura que el número total de aristas sea aproximadamente el 50 % del total de aristas posibles en un grafo completo. Luego se seleccionaron aristas del grafo según el porcentaje especificado (20 %, 50 % o 80 %). Estas aristas se seleccionaron aleatoriamente se les asignó el valor −1, respetando la cantidad determinada.

- **Conjunto intervalo unitario:** Este conjunto consta de 36 grafos los cuales tienen como subgrafo un grafo positivo de intervalos unitarios. El tamaño de los grafos corresponden a 100 vértices. Cada subgrafo se genera con valores de densidad denotado por ϵ correspondientes a $1/8, 1/2$ y $3/4$. Además, cada grafo incluye una cantidad fija de aristas negativas igual al 50 % del total de aristas.

  Para este conjunto, primero se generó un grafo de intervalos unitarios para la combinación de número de vértices (n) y valor de densidad (ϵ). El valor de ϵ define la longitud de los intervalos y, por ende, las conexiones entre los vértices, asegurando que el grafo cumpla las propiedades de un grafo de intervalos unitarios. Una vez generado el grafo base, se agregaron aristas negativas según el parámetro dado (20 %, 50 % o 80 % de las aristas totales). Las aristas negativas se agregaron seleccionándolas aleatoriamente entre las aristas no existentes, asegurando que no reemplazaran las aristas positivas originales.

5.1.2. Conjuntos de evaluación

- **Conjunto completo:** Este conjunto es similar al conjunto completo del conjunto de experimentación, con diferencia que cuenta con variedad de tamaños de grafos. En este caso, el conjunto está compuesto por 108 grafos completos. El tamaño de estos grafos varía de 10 a 230 en pasos de 20 vértices. Cada grafo cumple con que la cantidad de sus aristas negativas corresponde al 20 %, 50 % o 80 % del total de las aristas.
- **Conjunto aleatorio:** Este conjunto es similar al conjunto aleatorio del conjunto de experimentación con diferencia en la cantidad de aristas existentes. En este caso, el conjunto está compuesto por 117 grafos. El tamaño de los grafos varía de 10 a 250 vértices, en pasos de 20 vértices. La cantidad total de aristas varía en 20 %, 50 % y 80 % del total de las aristas posibles en el grafo. Luego, la cantidad de aristas negativas varía en 20 %, 50 % y 80 % del total de aristas existentes.
- **Conjunto intervalo unitario:** Al igual que los conjuntos anteriores, este conjunto es similar al conjunto Intervalo Unitario del conjunto de experimentación, con diferencia en la cantidad de aristas negativas que se

agregan. Este conjunto está compuesto por 117 grafos de intervalos unitarios. El tamaño de los grafos varía de 10 a 250 vértices, en pasos de 20 vértices. Para cada combinación de tamaño de grafo y parámetros de generación, se crean 3 grafos. Los parámetros incluyen valores de densidad denotado por ϵ los cuales varían entre $1/8, 1/2$ y $3/4$, y porcentajes de aristas negativas 20 %,50 % y 80 %.

5.2. Experimentos preliminares

En esta sección, se presentan los resultados obtenidos de los experimentos realizados para determinar los mejores parámetros para el algoritmo genético. Los parámetros considerados son: tipo de población, tamaño de población y función de mutación.

Para el tipo de población inicial, se consideran dos opciones: una generada por t-SNE y otra generada directamente por algoritmo genético. Para generar la población por medio de t-SNE, es necesario transformar los datos del grafo a una matriz de distancia.

Una matriz de distancia es una matriz que muestra las distancias entre un conjunto de puntos en un espacio determinado. Cada elemento (i, j) de la matriz representa la distancia entre el punto i y el punto j. Dado que es una distancia, cuanto menor sea el valor, más cerca estarán estos dos puntos.

Para facilitar el trabajo, primero se transforman los datos a una matriz de adyacencia y, a partir de esta, se crea la matriz de distancia. Sea una matriz de adyacencia de tamaño $n \times n$, donde los posibles valores son:

- 1: cuando existe una arista positiva.
- 0: cuando no hay aristas.
- -1: cuando la arista es negativa.

A partir de esta matriz de adyacencia, se generan seis tipos de matrices de distancias, que se presentan en la Tabla 5.1. Estos tipos de matrices de distancias se diferencian en cómo se asignan los valores de distancia según los valores de la matriz de adyacencia.

Tipo Matriz de distancia	Valor para $-\mathbf{1}$	Valor para $\mathbf{0}$	Valor para $\mathbf{1}$
Tipo 1	$\frac{n}{2}$	1	1
Tipo 2	n	1	1
Tipo 3	$\frac{n}{2}$	$\frac{n}{2}$	1
Tipo 4	n	$\frac{n}{2}$	1
Tipo 5	$\frac{n}{2}$	n	1
Tipo 6	n	n	1

Tabla 5.1: Valores de la matriz de distancias según la matriz de adyacencia.

Así los tipos de población posibles son siete, las seis mencionadas anteriormente más la generada por el algoritmo genético, la cual se llamara **Tipo Random**.

El otro parámetro a evaluar es el tamaño de población, es decir, la cantidad de individuos iniciales a mejorar para encontrar el óptimo. Para esto, se consideraron los tamaños 5, 10, 25, 50 y 100. A mayor cantidad de individuos en la población, mayor es la variabilidad y, a su vez, el costo computacional, dado que para pasar de una generación a otra se tiene que explorar muchos individuos.

El último parámetro es la función de mutación. Se consideraron dos funciones. Dado que se está trabajando con Python, la primera función a considerar y la que mejor que se adecuaba para este trabajo es la llamada *mutShuffleIndexes*. Esta función pertenece a la biblioteca *DEAP*. La operación que realiza esta función es mutar una lista intercambiando aleatoriamente algunos de sus elementos. Cada posición de la lista tiene una probabilidad de ser seleccionada para el intercambio, lo que introduce variabilidad en los individuos de un algoritmo genético. Por ejemplo, si se aplica a $[1, 2, 3, 4, 5]$ con probabilidad 0.5, algunos valores cambiarán de posición al azar en cada ejecución. La segunda función a considerar es SwapAndInsert, la cual se definió en la sección 4.2.2. Además de los parámetros mencionados, hay otros más los cuales se dejaron por defecto como los recomendaba la librería.

Experimentos: Al momento de realizar los experimentos, el conjunto completo y el conjunto aleatorio se dividieron en tres subconjuntos según la cantidad de aristas negativas. El primero corresponde a los grafos con una cantidad de aristas negativas del 20 %, el segundo subconjunto corresponde a los grafos con una cantidad de aristas negativas del 50 %, y el tercer subconjunto corresponde a los grafos con una cantidad de aristas negativas del 80 %.

En el caso del conjunto intervalo unitario, también se dividió en tres subconjuntos,

el criterio de la división fue según el valor de densidad de las aristas positivas. El primer subconjunto corresponde a los grafos con densidad 1/8 de aristas positivas, el segundo subconjunto corresponde los grafos con densidad 1/2 de aristas positivas y finalmente el tercer subconjunto corresponde a los grafos con densidad 3/4 de aristas positivas.

Estos subconjuntos fueron elegidos para cubrir una amplia gama de escenarios y evaluar cómo afectan el rendimiento del algoritmo.

Se realizaron dos experimentos con el fin de determinar los parámetros mencionados: tipo de población, tamaño de población y función de mutación. El primer experimento se llevó a cabo para determinar qué función de mutación utilizar. Para ello, se consideraron dos versiones del algoritmo genético: AG1 y AG2. AG1 tiene la función de mutación mutShuffleIndexes, que reordena aleatoriamente los índices de la solución, mientras que AG2 utiliza SwapAndInsert, que combina intercambios e inserciones para generar nuevas soluciones. Estas funciones fueron elegidas para evaluar cómo diferentes estrategias de mutación impactan en la exploración del espacio de soluciones.

Los tamaños de población elegidos para realizar esta prueba son 100, 50 y 25. El tipo de población se elige de acuerdo al tipo de conjunto utilizado. Dado que para el conjunto completo no es necesario considerar los seis tipo de población generados por t-SNE descrito anteriormente.

El experimento consiste en evaluar cada grafo de los subconjuntos utilizando AG1 y AG2. Se fijó el número de generaciones de los algoritmos genéticos en 100. Esta elección se debe a que, al trabajar con grafos de tamaño de 100 vértices, aumentar el número de generaciones significativamente incrementaría el costo computacional. Por lo tanto, el valor obtenido en la generación 100 es el utilizado para realizar las comparaciones. Se comparan los promedios a nivel de subconjunto y se determina cuál de los algoritmos genéticos tiene un valor menor. Además, se realiza un conteo de cuál algoritmo alcanza el valor más bajo para cada grafo.

Los resultados obtenidos se resumieron en tablas, las cuales se presentan en las subsecciones siguientes, para una mejor comprensión. Las tablas presentan los resultados del rendimiento de las dos versiones del algoritmo genético, AG1 y AG2, utilizando diferentes tamaños de población y tipos de población inicial (Tipo PI). También, se presentan los valores promedio de los resultados obtenidos en

cada experimento para cada versión del algoritmo (Promedio AG1 y Promedio AG2). Además, se muestra el número de veces que cada método coincide con la mejor solución encontrada en el experimento correspondiente (#Best AG1 y #Best AG2). Los valores representados en negritas en estas tablas representan el menor promedio entregado entre AG1 y AG2.

El segundo experimento se relaciona con la determinación del tipo de población inicial y el tamaño de población. Los tamaños de población considerados son 100, 50, 25, 10 y 5. Los tipos de población dependen del tipo de conjunto a usar. Al igual que el primer experimento, este se realiza por separado en cada subconjunto del conjunto completo.

Para realizar este experimento, en lugar de fijar las generaciones, se fijaron los tiempos de ejecución. Esto se debe a que al disminuir el tamaño de población (tamaños menores a 100), los algoritmos son más rápidos al recorrer las 100 generaciones. Así, el tiempo de ejecución fijado es de 60 segundos. Se elige este tiempo de ejecución tomando en cuenta los experimentos realizados por [27], para equiparar los tiempos al momento de comparar los rendimientos.

El experimento consiste en evaluar cada grafo de los subconjuntos utilizando el algoritmo genético seleccionado en el primer experimento, variando el tamaño de la población y el tipo de población. Se selecciona el valor entregado después de 60 segundos en cada grafo y luego se calcula el promedio según la combinación de tamaño de población y tipo de población.

Estos resultados también se presentan en tablas en las subsecciones siguientes. Para apreciar mejor los valores, se resaltan de la siguiente forma: el menor promedio es considerado el mejor y se resalta en negrita, el segundo valor más bajo se resalta con doble línea y el tercer valor se resalta con una línea.

A continuación, se presentan cuatro subsecciones. Las tres primeras presentan los resultados de los experimentos. En la última subsección, se presentan la elección de los parámetros que presentan el mejor Algoritmo Genético.

5.2.1. Experimentos sobre el conjunto completo

En esta subsección se presentan los resultados realizados sobre el conjunto completo de los conjuntos de experimentación. Una vez divido los subconjuntos, el tipo de población a considerar son el Tipo 1, Tipo 2 y Tipo Random. No se consideran otros tipos de población porque, al trabajar con grafos completos, no hay valores 0 en la matriz de adyacencia (los 0 en la diagonal no se consideran, ya que no se contabilizan los bucles).

En la Tabla 5.2 se puede apreciar que los valores de los promedios de AG1 y AG2 no presentan una diferencia muy significativa, pero en todos los casos AG2 es quién tiene el promedio más bajo. Esto se puede deber a que AG2 es quién mayormente encuentra valores más bajos para cada grafo del subconjunto en estudio.

El resultado anterior se puede observar tanto en la tabla 5.3, para el subconjunto con cantidad de aristas negativas de 50 %, como en la tabla 5.4, para el subconjunto con cantidad de aristas negativas de 80 %, por lo tanto se selecciona AG2 como la versión a usar para lo que sigue de experimento.

Tamaño Población	Tipo PI	Promedio AG1	Promedio AG2	# Best AG1	# Best AG2
100	Tipo 1	43873.58	**42941.33**	1	11
	Tipo 2	44033.08	**43043.67**	2	10
	Random	43946.58	**42939.50**	0	12
50	Tipo 1	44675.00	**43546.92**	0	12
	Tipo 2	44633.00	**43470.50**	0	12
	Random	44624.17	**43717.58**	1	11
25	Tipo 1	45172.25	**43900.33**	0	12
	Tipo 2	45379.00	**44029.50**	0	12
	Random	45249.00	**44056.58**	0	12

Tabla 5.2: Comparación de rendimiento entre AG1 y AG2 para grafos del conjunto completo de los conjuntos de experimentación, con un 20 % de aristas negativas. En negrita se aprecia los promedios más bajos entre AG1 y AG2.

Tamaño Población	Tipo PI	Promedio AG1	Promedio AG2	# Best AG1	# Best AG2
100	Tipo 1	69237.08	**68102.50**	0	12
	Tipo 2	69465.67	**67910.42**	0	12
	Random	69553.75	**68114.59**	0	12
50	Tipo 1	70076.25	**68123.33**	0	12
	Tipo 2	70450.17	**68459.75**	0	12
	Random	70203.33	**68314.33**	0	12
25	Tipo 1	71645.00	**69285.42**	0	12
	Tipo 2	71418.92	**69359.33**	0	12
	Random	71135.83	**69432.25**	1	11

Tabla 5.3: Comparación de rendimiento entre AG1 y AG2 para grafos del conjunto completo de los conjuntos de experimentación, con un 50 % de aristas negativas. En negrita se aprecia los promedios más bajos entre AG1 y AG2.

Tamaño Población	Tipo PI	Promedio AG1	Promedio AG2	# Best AG1	# Best AG2
100	Tipo 1	40865.25	**39634.25**	0	12
	Tipo 2	40902.08	**39603.41**	1	11
	Random	40623.67	**39249.67**	0	12
50	Tipo 1	42234.67	**40488.17**	1	11
	Tipo 2	41426.08	**39947.25**	0	12
	Random	41603.25	**39855.92**	1	11
25	Tipo 1	42566.92	**41001.67**	1	11
	Tipo 2	42706.75	**40555.75**	0	12
	Random	42816.58	**40922.83**	1	11

Tabla 5.4: Comparación de rendimiento entre AG1 y AG2 para grafos del conjunto completo de los conjuntos de experimentación, con un 80 % de aristas negativas. En negrita se aprecia los promedios más bajos entre AG1 y AG2.

Con la función de mutación fijada, falta a determinar el tipo de población inicial y el tamaño de población.

En las tablas 5.5, 5.6 y 5.7 se observa que, según los promedios calculados, cuando se utiliza una población inicial de Tipo Random y un tamaño de población de 25 ,10 o 5, se tiende a tener buenos resultados. En dos casos se repite que la población inicial de Tipo 1 más una población de tamaño 5, también entrega un buen resultado.

Si solo se consideran los tipos de población que tienen como entrada los datos procesados por t-SNE, la combinación de tipo de población Tipo 1 y tamaño de población de 5 es la que entrega los mejores resultados.

Tamaño Población	Tipo 1	Tipo 2	Tipo Random
100	44399.17	44553.17	43283.42
50	43296.00	43322.00	41767.17
25	43119.00	43103.67	41153.00
10	42904.42	43115.42	41289.83
5	**41020.75**	4151500	41144.08

Tabla 5.5: Comparación del rendimiento de AG2 según el tamaño de la población y el tipo de población para grafos del conjunto completo del conjunto de experimentación, con un 20 % de aristas negativas. Se aprecia que el valor más bajo es el obtenido por el tamaño de población 5 y tipo de población Tipo 1.

Tamaño Población	Tipo 1	Tipo 2	Tipo Random
100	70142.25	70158.25	68244.33
50	69164.83	69363.00	66175.17
25	68447.42	68708.00	64955.08
10	68181.50	68382.83	**64482.08**
5	64774.25	65138.92	64825.42

Tabla 5.6: Comparación del rendimiento de AG2 según el tamaño de la población y el tipo de población para grafos del conjunto completo del conjunto de experimentación, con un 50 % de aristas negativas. Se aprecia que el valor más bajo es el obtenido por el tamaño de población 10 y tipo de población Tipo Random.

Tamaño Población	Tipo 1	Tipo 2	Tipo Random
100	42065.67	41876.00	39321.25
50	40273.17	40785.08	38200.08
25	40196.58	40213.83	36876.25
10	39941.42	39943.00	**36692.00**
5	37092.75	36868.00	37183.75

Tabla 5.7: Comparación del rendimiento de AG2 según el tamaño de la población y el tipo de población para grafos del conjunto completo del conjunto de experimentación, con un 80 % de aristas negativas. Se aprecia que el valor más bajo es el obtenido por el tamaño de población 10 y tipo de población Tipo Random.

5.2.2. Experimentos sobre el conjunto aleatorio

En esta subsección, se presentan los resultados obtenidos sobre el conjunto aleatorio de los conjuntos de experimentación. Este conjunto se generó para evaluar el rendimiento de los algoritmos genéticos en escenarios menos estructurados y más variados que los del conjunto completo.

Se realizaron los dos experimentos presentados anteriormente. Para este conjunto, conjunto aleatorio, los tipos de población elegidos son: Tipo 1, Tipo 2, Tipo 3, Tipo 4, Tipo 5, Tipo 6 y Tipo Random. En comparación al conjunto completo, el conjunto Aleatorio tiene grafos los cuales no hay aristas que unan todos los vértices, por lo cual, la matriz de adyacencia cuanta con valores 0.

En Tabla 5.8 ,se observa que el comportamiento de los dos algoritmos es similar al obtenido en la sección anterior, en el sentido de que los valores promedios no tienen una diferencia significativa. Se repite el comportamiento de que AG2 mayormente alcanza los valores más bajos para cada grafo del subconjunto con una cantidad de aristas negativas del 20 %.

El experimento se realizó para los dos subconjuntos restantes, obteniendo resultados similares al caso anterior. En Tabla 5.9, del subconjunto con una cantidad de aristas negativas del 50 % y la Tabla 5.10, del subconjunto con una cantidad de aristas negativas del 80 %, se resume lo obtenido en los experimentos. Así, se selecciona AG2 para seguir con los experimentos en esta sección.

Tamaño Población	Tipo PI	Promedio AG1	Promedio AG2	# Best AG1	# Best AG2
100	Tipo 1	10244.25	**9908.50**	1	11
	Tipo 2	10198.58	**9861.75**	0	12
	Tipo 4	10235.58	**9838.92**	0	12
	Tipo 4	10180.17	**9755.08**	2	10
	Tipo 5	10219.83	**9835.58**	0	12
	Tipo 6	10216.58	**9735.25**	0	12
	Random	10165.33	**9815.33**	0	12
50	Tipo 1	10353.50	**9975.00**	2	10
	Tipo 2	10466.50	**9997.42**	0	12
	Tipo 4	10392.50	**9955.67**	1	11
	Tipo 4	10442.50	**9965.17**	0	12
	Tipo 5	10353.83	**9957.92**	0	12
	Tipo 6	10453.08	**9868.08**	0	12
	Random	10449.83	**9956.42**	0	12
25	Tipo 1	10777.25	**10209.33**	0	12
	Tipo 2	10634.58	**10116.33**	1	11
	Tipo 4	10731.83	**10164.08**	0	12
	Tipo 4	10686.67	**10165.25**	0	12
	Tipo 5	10629.92	**10120.42**	0	12
	Tipo 6	10850.50	**10253.67**	0	12
	Random	10559.67	**10123.67**	1	11

Tabla 5.8: Comparación de rendimiento entre AG1 y AG2 para grafos del conjunto aleatorio de los conjuntos de experimentación, con un 20 % de aristas negativas. En negrita se aprecia los promedios más bajos entre AG1 y AG2.

Tamaño Población	Tipo PI	Promedio AG1	Promedio AG2	# Best AG1	# Best AG2
100	Tipo 1	15671.33	**15111.92**	1	11
	Tipo 2	15791.83	**15279.25**	1	11
	Tipo 3	15679.50	**15131.83**	0	12
	Tipo 4	15657.67	**15163.33**	1	11
	Tipo 5	15613.58	**15168.33**	0	12
	Tipo 6	15709.75	**15256.25**	1	11
	Random	15740.25	**15170.00**	0	12
50	Tipo 1	16097.83	**15423.58**	0	12
	Tipo 2	15913.67	**15365.33**	1	11
	Tipo 3	16070.58	**15401.41**	1	11
	Tipo 4	16114.25	**15337.75**	0	12
	Tipo 5	16063.67	**15460.25**	1	11
	Tipo 6	15920.42	**15409.50**	0	12
	Random	15980.17	**15226.50**	0	12
25	Tipo 1	16550.58	**15821.50**	0	12
	Tipo 2	16358.67	**15603.58**	0	12
	Tipo 3	16520.00	**15556.17**	0	12
	Tipo 6	16432.33	**15709.67**	0	12
	Tipo 5	16290.00	**15589.83**	0	12
	Tipo 6	16325.83	**15669.25**	0	12
	Random	16518.25	**15695.83**	0	12

Tabla 5.9: Comparación de rendimiento entre AG1 y AG2 para grafos del conjunto aleatorio de los conjuntos de experimentación, con un 50 % de aristas negativas. En negrita se aprecia los promedios más bajos entre AG1 y AG2.

Tamaño Población	Tipo PI	Promedio AG1	Promedio AG2	# Best AG1	# Best AG2
100	Tipo 1	9009.50	**8391.58**	1	11
	Tipo 2	8925.58	**8456.25**	0	12
	Tipo 3	8933.42	**8419.00**	1	11
	Tipo 4	8986.92	**8492.00**	0	12
	Tipo 5	8970.25	**8366.42**	0	12
	Tipo 6	8950.58	**8441.00**	0	12
	Random	8941.67	**8454.00**	0	12
50	Tipo 1	9254.75	**8681.83**	1	11
	Tipo 2	9344.08	**8534.67**	0	12
	Tipo 3	9318.67	**8538.50**	0	12
	Tipo 4	9435.83	**8537.50**	0	12
	Tipo 5	9298.92	**8644.42**	0	12
	Tipo 6	9407.08	**8666.67**	0	12
	Random	9261.08	**8670.50**	0	12
25	Tipo 1	9501.67	**8894.25**	1	11
	Tipo 2	9624.25	**8877.92**	0	12
	Tipo 3	9649.00	**8952.00**	0	12
	Tipo 4	9676.08	**9024.33**	0	12
	Tipo 5	9727.83	**8902.00**	0	12
	Tipo 6	9651.92	**8860.42**	0	12
	Random	9722.58	**8873.42**	0	12

Tabla 5.10: Comparación de rendimiento entre AG1 y AG2 para grafos del conjunto aleatorio de los conjuntos de experimentación, con un 80 % de aristas negativas. En negrita se aprecia los promedios más bajos entre AG1 y AG2.

Con la función de mutación seleccionada, falta definir el tipo de población inicial y el tamaño de población. Los tipos de población usados para el segundo experimento son Tipo 1, Tipo 2, Tipo 4 y Tipo Random. Se consideraron solo estos tipos dado que fueron los que entregaban más veces valores bajos en comparación al resto de tipos.

En las tablas 5.11, 5.12 y 5.13 se observa que, según los promedios calculados, cuando se utiliza un tamaño de población de 10, se obtienen los resultados más bajos en los tres subconjuntos. Además, se observa que un subconjunto la población de Tipo 1 tiene el promedio más bajo y en otro caso es el segundo más bajo. El Tipo 2 es dos veces el segundo promedio más bajo. El Tipo 4 se observa una vez como el promedio más bajo y una vez como el tercero más bajo. Por último, el Tipo Random es una vez el que entrega mejor promedio, mientras que dos veces es el que entrega segundo mejor promedio.

Tamaño Población	Tipo 1	Tipo 2	Tipo 4	Random
100	10210.50	10237.75	10148.25	9842.17
50	9407.33	9450.75	9535.83	9339.42
25	9216.75	9192.75	9233.33	9122.75
10	**9011.17**	9096.50	9075.25	9064.66
5	9167.33	9193.75	9133.00	9140.92

Tabla 5.11: Comparación del rendimiento de AG2 según el tamaño de la población y el tipo de población para grafos del conjunto aleatorio del conjunto de experimentación, con un 20 % de aristas negativas. Se aprecia que el valor más bajo es el obtenido por el tamaño de población 10 y tipo de población Tipo 1.

Tamaño Población	Tipo 1	Tipo 2	Tipo 4	Random
100	15960.83	15881.25	15842.33	15154.83
50	14681.25	14787.17	14649.92	14624.67
25	14307.67	14261.25	14235.50	14166.58
10	14111.75	14110.75	**14013.33**	14099.33
5	14312.00	14267.42	14207.25	14198.25

Tabla 5.12: Comparación del rendimiento de AG2 según el tamaño de la población y el tipo de población para grafos del conjunto aleatorio del conjunto de experimentación, con un 50 % de aristas negativas. Se aprecia que el valor más bajo es el obtenido por el tamaño de población 10 y tipo de población Tipo 4.

Tamaño Población	Tipo 1	Tipo 2	Tipo 4	Random
100	9252.75	9043.66	9112.58	8440.00
50	8103.666	8159.00	8218.00	7911.25
25	7689.75	7661.75	7707.00	7604.42
10	7464.08	7488.08	7551.17	**7438.83**
5	7588.83	7594.50	7515.92	7678.75

Tabla 5.13: Comparación del rendimiento de AG2 según el tamaño de la población y el tipo de población para grafos del conjunto aleatorio del conjunto de experimentación, con un 80 % de aristas negativas. Se aprecia que el valor más bajo es el obtenido por el tamaño de población 10 y tipo de población Tipo Random.

5.2.3. Experimentos sobre el conjunto intervalo unitario

En esta subsección, se presentan los experimentos y resultados realizados sobre un conjunto intervalo unitario de los conjuntos de experimentación. Este conjunto se generó para evaluar el rendimiento de los algoritmos genéticos en escenarios donde las relaciones entre los vértices están definidas por intervalos unitarios, por lo cual se sabe cual es su valor óptimo. Aunque para efecto de los experimentos se seguirá considerando el valor más bajo entregado por AG1 y AG2 para decir cuál de los dos llego a un mejor óptimo.

Para los tipos de población inicial se consideraron los tipos de población: Tipo 1, Tipo 2, Tipo 3, Tipo 4, Tipo 5, Tipo 6 y Tipo Random.

En la tabla 5.14, se observa que el comportamiento de los dos algoritmos es similar al obtenido en las secciones anteriores, en el sentido de que los valores promedios no tienen una diferencia significativa. Se repite el comportamiento de que AG2 mayormente alcanza los valores más bajos para cada grafo del subconjunto con una cantidad de aristas negativas del 20 %.

El experimento se realizó para los dos subconjuntos restantes, obteniendo resultados similares al caso anterior. En las tablas 5.15 y 5.16, se resume lo obtenido en los experimentos para los subconjuntos con una cantidad de aristas negativas del 50 % y 80 %, respectivamente. Así, se selecciona AG2 para seguir con los experimentos en esta sección.

Tamaño Población	Tipo PI	Promedio AG1	Promedio AG2	# Best AG1	# Best AG2
100	Tipo 1	10649.92	**8133.83**	1	11
	Tipo 2	11407.17	**9322.25**	2	10
	Tipo 3	10814.08	**9105.25**	1	11
	Tipo 4	11439.00	**8893.83**	0	12
	Tipo 5	11081.25	**9209.25**	2	10
	Tipo 6	11291.58	**8929.83**	1	11
	Random	11123.25	**8664.33**	0	12
50	Tipo 1	13164.17	**10349.50**	0	12
	Tipo 2	12762.50	**10339.83**	1	11
	Tipo 3	12503.33	**10208.66**	1	11
	Tipo 4	13564.83	**10030.91**	1	11
	Tipo 5	13738.83	**10531.25**	0	12
	Tipo 6	13248.92	**9132.92**	1	11
	Random	14257.83	**10121.83**	0	12
25	Tipo 1	15758.00	**11934.08**	0	12
	Tipo 2	15375.33	**11614.75**	1	11
	Tipo 3	16362.33	**11871.58**	0	12
	Tipo 4	15897.17	**11834.33**	0	12
	Tipo 5	15037.67	**11878.42**	0	12
	Tipo 6	15221.33	**12163.92**	0	12
	Random	15134.83	**12151.92**	1	11

Tabla 5.14: Comparación de rendimiento entre AG1 y AG2 para grafos del conjunto intervalo unitario de los conjuntos de experimentación, con densidad del 20 % de aristas positivas. En negrita se aprecia los promedios más bajos entre AG1 y AG2.

Tamaño Población	Tipo PI	Promedio AG1	Promedio AG2	# Best AG1	# Best AG2
25	Tipo 1	10823.92	**6380.67**	0	12
	Tipo 2	11645.08	**6631.08**	0	12
	Tipo 3	11163.08	**6928.58**	0	12
	Tipo 4	10891.00	**6914.83**	0	12
	Tipo 5	10843.08	**6847.50**	0	12
	Tipo 6	11645.67	**6289.75**	0	12
	Random	10673.08	**6542.58**	0	12
50	Tipo 1	7830.83	**4723.17**	0	12
	Tipo 2	8269.00	**5012.67**	1	11
	Tipo 3	8126.25	**4949.92**	0	12
	Tipo 4	8348.17	**5311.75**	0	12
	Tipo 5	8116.67	**4710.75**	0	12
	Tipo 6	8897.17	**5091.42**	0	12
	Random	7873.17	**5327.92**	0	12
100	Tipo 1	5866.08	**4120.92**	0	12
	Tipo 2	6030.00	**4135.58**	0	12
	Tipo 3	6675.58	**4334.83**	0	11
	Tipo 4	6289.17	**4310.42**	0	12
	Tipo 5	6043.33	**4082.42**	0	12
	Tipo 6	6310.92	**4317.67**	0	12
	Random	6299.92	**3925.67**	0	12

Tabla 5.15: Comparación de rendimiento entre AG1 y AG2 para grafos del conjunto intervalo unitario de los conjuntos de experimentación, con densidad del 50 % de aristas positivas. En negrita se aprecia los promedios más bajos entre AG1 y AG2.

Tamaño Población	Tipo PI	Promedio AG1	Promedio AG2	# Best AG1	# Best AG2
25	Tipo 1	2742.58	**1716.00**	1	11
	Tipo 2	2545.33	**2031.00**	2	10
	Tipo 3	2836.67	**1724.25**	1	11
	Tipo 4	2917.42	**2056.92**	1	11
	Tipo 5	3083.75	**1806.33**	0	12
	Tipo 6	3035.75	**1819.00**	0	12
	Random	2920.17	**2023.00**	1	11
50	Tipo 1	2085.92	**1402.25**	2	10
	Tipo 2	1978.92	**1253.75**	0	12
	Tipo 3	2101.42	**1210.00**	1	11
	Tipo 4	2103.00	**1020.75**	0	12
	Tipo 5	1992.00	**1236.83**	0	12
	Tipo 6	1829.08	**1164.83**	0	12
	Random	2122.00	**1208.83**	1	11
100	Tipo 1	1500.42	**867.83**	0	12
	Tipo 2	1500.42	**874.92**	0	12
	Tipo 3	1551.58	**882.17**	0	12
	Tipo 4	1497.92	**918.25**	1	11
	Tipo 5	1561.25	**940.83**	0	12
	Tipo 6	1548.42	**898.83**	0	12
	Random	1347.00	**947.75**	1	11

Tabla 5.16: Comparación de rendimiento entre AG1 y AG2 para grafos del conjunto intervalo unitario de los conjuntos de experimentación, con densidad del 80 % de aristas positivas. En negrita se aprecia los promedios más bajos entre AG1 y AG2.

Siguiendo con el segundo experimento. Los tipos de población seleccionados son Tipo 1, Tipo 2, Tipo 3, Tipo 5 y Tipo Random. Se consideraron solo estos tipos dado que fueron los que entregaban más veces valores bajos en comparación con el resto de tipos. Al igual que el primer experimento, este experimento se realiza por separado en cada subconjunto del conjunto intervalo unitario.

En las tablas 5.17, 5.18 y 5.19 se observa un comportamiento similar al los subconjuntos del conjunto aleatorios. De los promedios se observa que, cuando se utiliza un tamaño de población de 10, se obtienen los resultados más bajos en los tres subconjuntos del conjunto intervalo unitario. También, se observa que un subconjunto la población de Tipo 1 con un tamaño de población de 10, tiene el promedio más bajo. El Tipo 2 en un subconjunto, para los grafos del conjunto intervalo unitario de densidad 1/2, entrega dos de los promedios más bajos, segundo y tercero valores más bajos. El Tipo 3 solo se observa una vez como el segundo promedio más bajo. El tipo población Tipo 5 tampoco resalta tanto, con un tercer buen promedio en un subconjunto. Por último, el Tipo Random tiene un mejor rendimiento de todos los tipo. Dos veces entrega mejor promedio, con tamaño de población 10, y una vez es el que entrega segundo mejor promedio.

Tamaño Población	Tipo 1	Tipo 2	Tipo 3	Tipo 5	Random
100	12248.67	13216.92	12006.83	12109.67	7794.08
50	5968.83	6467.75	6060.92	5388.50	4790.50
25	3413.17	2256.42	3326.92	4600.50	2701.92
10	3039.83	2679.83	2764.17	3363.92	**1883.50**
5	4075.17	3790.58	2775.83	3359.67	3375.33

Tabla 5.17: Comparación del rendimiento de AG2 según el tamaño de la población y el tipo de población para grafos del conjunto intervalo unitario del conjunto de experimentación, con una densidad de 1/8 de aristas positivas. Se aprecia que el valor más bajo es el obtenido por el tamaño de población 10 y tipo de población Tipo Random.

Tamaño Población	Tipo 1	Tipo 2	Tipo 3	Tipo 5	Random
100	6284.83	6744.91	7156.58	6715.08	44796.25
50	1268.50	1110.08	1214.08	1169.67	691.50
25	232.83	240.33	226.58	235.33	171.08
10	118.67	119.17	109.91	110.42	**105.58**
5	836.50	914.97	814.33	728.08	607.50

Tabla 5.18: Comparación del rendimiento de AG2 según el tamaño de la población y el tipo de población para grafos del conjunto intervalo unitario del conjunto de experimentación, con una densidad de 1/2 de aristas positivas. Se aprecia que el valor más bajo es el obtenido por el tamaño de población 10 y tipo de población Tipo Random.

Tamaño Población	Tipo 1	Tipo 1	Tipo 2	Tipo 4	Random
100	1533.67	1633.17	1592.42	1601.75	928.00
50	136.75	148.67	129.83	153.50	108.33
25	34.42	41.08	38.17	43.42	22.67
10	**20.75**	42.75	44.67	68.00	53.33
5	224.08	298.08	266.67	134.75	200.17

Tabla 5.19: Comparación del rendimiento de AG2 según el tamaño de la población y el tipo de población para grafos del conjunto intervalo unitario del conjunto de experimentación, con una densidad de 3/4 de aristas positivas. Se aprecia que el valor más bajo es el obtenido por el tamaño de población 10 y tipo de población Tipo 1.

5.2.4. Resultados de experimentos preliminares

En esta subsección se seleccionan los parámetros que mejor rendimiento tienen para el algoritmo genético, para luego realizar la comparación de rendimiento con la heurística BVNS.

Es claro que la función de mutación que mejor rendimiento presenta es la función SwapAndInsert. Ahora falta fijar los parámetros de tamaño de población y tipo de población.

Para los dos parámetros restantes, se tiene que, debido a la variabilidad de los promedios obtenidos en el segundo experimento y la variabilidad de las posibles combinaciones de tamaño de población y tipo de población que tiene buen rendimiento por subconjunto, se utilizó en un sistema de puntuación por etapas, para decidir la mejor combinación de tamaño de población y tipo de población. Este sistema de puntuación es inspirado en *El sistema de puntuación de la fórmula 1*, donde por etapa, dependiendo del lugar de cada competidor se le asigna un puntaje [1].

Inicialmente se selecciona los tipos de población a usar para realizar la selección sobre qué combinación de tamaño de población y tipo de población tiene mejor rendimiento. De los tres conjuntos de experimentación, se tiene en común que los tipos de población que tienen buen rendimiento son Tipo 1, Tipo 2 y Tipo Random. Tomando solo los datos correspondientes de los resultados del segundo experimento de estos tipos de población, se realiza la elección. En este caso, las etapas son los distintos resultados por subconjuntos, y el sistema de elección es el siguiente:

- En cada etapa, se realiza una evaluación de todos los participantes.
- Los tres promedios más bajos en cada etapa reciben puntos, y los demás no obtienen puntos.
- Los puntos son, 3 puntos al primero, 2 puntos al segundo y 1 punto al tercero.

[1] Para mas información del sistema de puntos se puede ingresar a: https://www.caranddriver.com/es/formula-1/a44339500/puntos-formula-1-posiciones-bonus-tipo-carrera/
o https://es.motorsport.com/f1/news/sistema-puntos-posiciones-reparto-formula1/6505476/

- La combinación de tamaño de población y tipo de población con más puntos es elegido como los parámetros del algoritmo

Una vez realizada la elección, las tres combinaciones mejor puntuadas son:

- Con 20 puntos, la mejor combinación es cuando se tiene tamaño 10 y tipo de población Tipo Random.
- Con 11 puntos, la segunda mejor combinación es cuando se tiene tamaño 10 y tipo de población Tipo 1.
- Con 7 puntos, la tercera mejor combinación es cuando se tiene tamaño 10 y tipo de población Tipo 2.

Con lo anterior, se tiene que el algoritmo genético que mejor se desempeña, es aquel que tiene población inicial 10, tipo de población Random, es decir, la población generada por el algoritmo Genético, y la función de mutación SwapAndInsert.

A modo de no descartar lo realizado por t-SNE, es que para efecto de comparación también se va a utilizar un algoritmo genético con pre-proceso. Así, se tiene otro algoritmo genético con tamaño de población 10, población inicial Tipo 1, que es un tipo generada por el t-SNE, y la función de mutación SwapAndInsert.

De esta manera, los dos algoritmos a comparar con el algoritmo BVNS son AG, para referirse al Algoritmo genético sin pre-proceso, y el algoritmo TSNE-AG, para referirse al algoritmo genético con pre-proceso.

Comparación

En este capítulo se presenta dos secciones donde se realizan comparación de los algoritmo. En la primera sección se realizará la comparación de los dos algoritmos seleccionados en los experimentos del capitulo anterior con BVNS para la primera métrica ET a optimizar. En la segunda sección, se aplica la misma comparación que en la primera sección modificando la función objetivo o la métrica a optimizar por la métrica EM y EV.

BVNS se implementó con los parámetros que aseguraban el mejor rendimiento según lo descrito en [27]. Este algoritmo cuenta con un pre-proceso constructivo descrito en la sección 4.1.3. El paso Shake a usar consiste en movimientos de intercambio. También se emplea LS1, basado en movimientos de inserción y la vecindad límite se fija en $k_{max} = 4$. Por último, el tiempo límite fijado t_{max} se ajusta para que coincida con algoritmos genéticos.

Para esta comparación, se consideran los conjuntos de evaluación, los cuales son, Conjunto Completo, Conjunto Aleatorio y Conjunto de Intervalo Unitario descritos en 5.1.1. No se realiza la división de subconjuntos según la cantidad de aristas negativas, sino que se realizan sobre los conjuntos completos.

6.1. Comparación para la optimización de ET

En la Tabla 6.1 se presentan los resultados de los experimentos. Para los algoritmos se determinó como tiempo límite de ejecución de 60 segundos para sea acorde a los tiempo de los experimentos realizados en [27]. A su vez se incluyen los tiempo de pre-procesos correspondientes (Algoritmo constructivo y t-SNE). Para

comparar el rendimiento se calculó la desviación porcentual promedio (Dev(%)) y se contabilizó el número de veces que cada método coincide con la mejor solución encontrada entre los tres algoritmos (#Best).

La desviación porcentual promedio es una medida que indica cuánto se desvían los resultados de un valor óptimo o promedio. Se calcula como la diferencia absoluta entre cada resultado y el valor óptimo, dividida por el valor óptimo, y luego se promedian estas desviaciones porcentuales. Sin embargo, si el valor óptimo es cero, esta medida no se puede calcular directamente debido a la división por cero. Por lo cual, cuando el óptimo es 0, si resultado es 0, se asigna como valor de desviación porcentual 0. En el caso que el resultado es distinto de 0, se le asigna como valor de desviación porcentual 100.

En la tabla 6.1, el cálculo de la desviación porcentual promedio para el conjunto intervalo unitario se consideró como óptimo la mejor solución encontrada entre los tres algoritmos. En cambio en la tabla 6.2, dado la propiedad que tiene el conjunto de intervalo unitario, se fijó como óptimo en todos los casos el valor 0.

Conjunto	**Algoritmo**	**Tiempo pre-proceso**	**Tiempo proceso**	Dev(%)	# **Best**
Completo	BVNS	38.29	60.09	1.90	24
	AG	-	60.05	1.68	45
	TSNE-AG	2.22	60.06	0.68	59
Aleatorio	BVNS	36.53	60.08	4.04	35
	AG	-	60.07	4.90	41
	TSNE-AG	2.83	60.08	4.08	63
Intervalo Unitario	BVNS	20.45	60.09	95.09	59
	AG	-	60.06	121.39	49
	TSNE-AG	2.82	60.07	574.82	58

Tabla 6.1: Comparación de los algoritmos para los conjuntos de evaluación, fijando el tiempo de ejecución de los algoritmos a 60 segundos.

De la tabla 6.1 se puede concluir que el algoritmo TSNE-AG es capaz de obtener un mayor número de # Best en dos conjuntos (Completo y Aleatorio), mientras que en el tercer conjunto (Intervalo Unitario) no hay una diferencia significativa con el algoritmo que mejor lo hace (difiere en 1 con BVNS). En cuanto en la desviación obtenida y analizando conjunto a conjunto, TSNE-AG es el que tiene mejor rendimiento en uno de los tres conjuntos (Completo), mientras que en los otros dos conjunto el que mejor lo hace es BVNS.

Se observa además que en el caso del Conjunto Intervalo Unitario, las desviaciones

presentan valores muy altos. Pese que BVNS y el TSNE-AG no difieren mucho en tener el mayor numero de # Best, el TSNE-AG presenta una desviación muy alta en comparación del que tiene menor desviación (BVNS). Recordar que para este caso no se consideraron como optimo los valores 0.

En la tabla 6.2, el conjunto de intervalo unitario, tiene fijado como óptimo en todos los casos el valor 0. De esta tabla se observa que TSNE-AG obtiene un mayor número de # Best, seguido por AG y en tercer lugar BVNS. La diferencia de estos valores no es muy significativa, por lo cual se puede pensar que son capaces de llegar al óptimo 0 en casi los mismos grafos, esto se puede deber al tamaño de las grafos. Mientras más numero de vértices tiene un grafo, más se le es difícil a los algoritmos llegar a un óptimo.

Conjunto	**Algoritmo**	**Tiempo pre-proceso**	**Tiempo proceso**	Dev(%)	**# Best**
Intervalo	BVNS	20.45	60.09	79.49	24
Unitario	AG	-	60.06	78.63	25
	TSNE-AG	2.82	60.07	76.92	27

Tabla 6.2: Comparación de los algoritmos sobre el conjunto de intervalo unitario considerando como valor óptimo 0 en todos los grafos, fijando el tiempo de ejecución de los algoritmos a 60 segundos

6.2. Comparación para las optimizaciones de EM y EV

Además de optimizar la métrica ET, también se presentan la optimización de dos métricas más, presentadas en la Definición 2.2, EM y ET.

En la tabla 6.3, se encuentra resumido lo obtenido al comparar el rendimiento de los algoritmos cuando la métrica a optimar corresponde a EM.

Se observa que el algoritmo TSNE-AG es capaz de obtener un mayor número de # Best en dos conjuntos (Completo y Aleatorio). En el tercer conjunto (Intervalo Unitario) el que tiene un mayor número de # Best el algoritmo BVNS. En cuanto en la desviación obtenida y analizando conjunto a conjunto, se repite la dinámica de la cantidad de # Best.

TSNE-AG es el que tiene mejor rendimiento en uno de los tres conjuntos (Completo), mientras que en los otros dos conjunto el que mejor lo hace es

BVNS. El TSNE-AG es el que tiene mejor valor en dos conjunto (Completo y Aleatorio) y el BVNS es el que tiene mejor rendimiento en el tercer conjunto (Intervalo Unitario).

Al igual que en la sección anterior, se hizo un análisis para los valores óptimos del conjunto Intervalo Unitario. En la tabla 6.4, se presentan los resultados con los valores óptimos igual 0. Se observa que para el número de # Best, los algoritmos AG y TSNE-AG, tienen la misma cantidad, donde esta cantidad es la mayor en comparación al BVNS. Los mismos son los que tiene la misma desviación. Mencionar que estos algoritmos llegaron 13 veces al óptimo de 117, lo cual es un número bajo y se ve reflejado en sus valores de desviación (88.89).

Conjunto	**Algoritmo**	**Tiempo pre-proceso**	**Tiempo proceso**	Dev(%)	# **Best**
Completo	BVNS	38.35	60.07	6.23	39
	AG	-	60.05	2.56	38
	TSNE-AG	2.35	60.06	2.40	53
Aleatorio	BVNS	35.08	60.07	10.54	38
	AG	-	60.08	4.77	49
	TSNE-AG	2.89	60.08	2.88	59
Intervalo Unitario	BVNS	19.26	86.08	95.09	60
	AG	-	60.06	201.77	38
	TSNE-AG	2.83	60.05	124.00	46

Tabla 6.3: Comparación de los algoritmos para los conjuntos de evaluación, fijando el tiempo de ejecución de los algoritmos a 60 segundos.

Conjunto	**Algoritmo**	**Tiempo pre-proceso**	**Tiempo proceso**	Dev(%)	# **Best**
Intervalo Unitario	BVNS	19.26	90.60	95.09	11
	AG	-	60.06	88.89	13
	TSNE-AG	2.83	60.05	88.89	13

Tabla 6.4: Comparación de los algoritmos sobre el conjunto de intervalo unitario considerando como valor óptimo 0 en todos los grafos, fijando el tiempo de ejecución de los algoritmos a 60 segundos.

Para el caso que la métrica a optimizar corresponde a la métrica EV, los resultados obtenidos se resumen en la tabla 6.5 y la tabla 6.6. De la primera tabla se puede concluir que para el conjunto Completo, los tres algoritmos tiene un desempeño notable, dado que los tres presenta una cantidad de # Best superior a 100 (siendo el total 108). El que mejor desempeño tiene en este conjunto es el algoritmo TSNE-AG tanto en el número de #Best como en el valor de la desviación. Los otros dos algoritmos también tiene un valor bajo en la desviación.

Para el caso del conjunto aleatorio, el que mejor desempeño presenta es el algoritmo AG, dado que presenta un mayor número de # Best y un valor de desviación más bajo que los otros dos algoritmos. El segundo algoritmo con buen rendimiento es TSNE-AG dado que no difiere mucho en la cantidad de # Best pero si difiere un poco más en la desviación.

Al trabajar sobre el conjunto de intervalo unitario, sin fijar los óptimos igual a 0, se obtiene que el algoritmo BVNS es quien tiene mayor cantidad de #Best. Para la desviación el mismo algoritmo, BVNS, es quien presenta el menor valor de desviación.

En la tabla 6.6 se observa que al fijar los óptimos igual a 0, cambia el rendimiento mencionado anteriormente. Los algoritmos AG y BVNS-AG tiene mejor rendimiento en cuanto la cantidad de #Best y el valor de la desviación. Al igual que en el caso de cuando se optimiza la métrica EM, la cantidad de #Best es bajo (15), mientras que el valor de la desviación es alta (87.18).

Conjunto	**Algoritmo**	**Tiempo pre-proceso**	**Tiempo proceso**	Dev(%)	**# Best**
Completo	BVNS	37.67	60.08	0.20	101
	AG	-	60.05	0.13	103
	TSNE-AG	2.36	60.06	0.10	104
Aleatorio	BVNS	36.03	60.07	7.43	80
	AG	-	60.07	0.71	99
	TSNE-AG	2.83	60.07	3.38	98
Intervalo Unitario	BVNS	19.78	60.06	95.09	88
	AG	-	60.05	121.39	48
	TSNE-AG	2.90	60.05	574.82	46

Tabla 6.5: Comparación de los algoritmos para los conjuntos de evaluación, fijando el tiempo de ejecución de los algoritmos a 60 segundos.

Conjunto	**Algoritmo**	**Tiempo pre-proceso**	**Tiempo proceso**	Dev(%)	**# Best**
Intervalo Unitario	BVNS	19.78	60.06	90.60	11
	AG	-	60.05	87.18	15
	TSNE-AG	2.90	60.05	87.18	15

Tabla 6.6: Comparación de los algoritmos sobre el conjunto de intervalo unitario considerando como valor óptimo 0 en todos los grafos, fijando el tiempo de ejecución de los algoritmos a 60 segundos.

Conclusión

En esta tesis se presentaron los Algoritmos Genéticos con el fin de implementarlos en la optimización de tres métricas (ET, EM y EV). De trabajos previos, se tenía conocimiento de que para la primera métrica ya existían algoritmos implementados con resultados satisfactorios. El algoritmo con mejor rendimiento era el algoritmo BVNS. Así, además de implementar los Algoritmos Genéticos, se buscaba mejorar el rendimiento del algoritmo BVNS.

Para realizar la comparación, se utilizaron dos variaciones de Algoritmos Genéticos. La diferencia entre estas dos variaciones es que una considera la población inicial generada por defecto del Algoritmo Genético (AG), mientras que la otra variación tiene como población inicial la generada por el algoritmo t-SNE.

De la comparación se puede destacar que, en general, el algoritmo TSNE-AG tiene mejor rendimiento en los conjuntos Completo y Aleatorio, considerando las tres métricas. Para el caso del conjunto Intervalo Unitario, en las tres métricas, el BVNS sigue siendo el de mejor rendimiento si no se consideran valores óptimos en todos los casos. Al considerar los valores óptimos como 0, cambia el algoritmo que mejor se desempeña: en la métrica ET, el algoritmo TSNE-AG es el que tiene mejor desempeño. En las otras dos métricas se observa que tanto el algoritmo AG como el algoritmo TSNE-AG son los de mejor rendimiento, con los mismos valores en # Best y desviación.

Cabe mencionar que, según los resultados preliminares, se esperaba que el algoritmo AG tuviera un mejor rendimiento que el algoritmo TSNE-AG, sin embargo, ocurrió lo contrario. Este resultado se puede deber a que los experimentos preliminares se realizaron con grafos de tamaño 100, mientras que la comparación se llevó a

cabo con grafos de tamaños que variaban entre 10 y 250. Quizás, en los grafos de mayor tamaño, el desempeño del algoritmo AG disminuye. Por otro lado, el algoritmo TSNE-AG comienza con una población inicial que es un buen punto de partida, gracias a la reducción de dimensionalidad proporcionada por t-SNE. Quizás este buen punto de partida permita al algoritmo TSNE-AG tener un mejor rendimiento que AG en los casos de mayor tamaño.

Bibliografía

[1] M. Batista, J. A. M. Pérez, and J. M. M. Vega. algoritmos genéticos. una visión práctica. *Números: Revista de didáctica de las matemáticas (Ejemplar dedicado a: Darwin)*, (71), 2009.

[2] S. L. Bezrukov, J. D. Chavez, L. H. Harper, M. Röttger, and U.-P. Schroeder. The congestion of n-cube layout on a rectangular grid. *Discrete Mathematics*, 213(1-3):13–19, 2000.

[3] P. Boominathan and A. Kanchan. Routing planning as an application of graph theory. *International journal of scientific & technology research*, 3: 61–66, 2014.

[4] C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM*, 16(9):575–577, 1973.

[5] M. Cygan, M. Pilipczuk, M. Pilipczuk, and J. O. Wojtaszczyk. Sitting closer to friends than enemies, revisited. *Theory of computing systems*, 56(2): 394–405, 2015.

[6] R. Diestel. *Graph Theory*. Springer, 2010.

[7] G. Ding and B. Oporowski. Some results on tree decomposition of graphs. *Journal of Graph Theory*, 20(4):481–499, 1995.

[8] A. Duarte, J. J. Pantrigo, E. G. Pardo, and N. Mladenovic. Multi-objective variable neighborhood search: an application to combinatorial optimization problems. *Journal of Global Optimization*, 63:515–536, 2015.

[9] A. Duarte, J. J. Pantrigo, E. G. Pardo, and J. Sánchez-Oro. Parallel variable neighbourhood search strategies for the cutwidth minimization problem. *IMA Journal of Management Mathematics*, 27(1):55–73, 2016.

[10] P. Formanowicz and K. Tanaś. A survey of graph coloring-its types, methods and applications. *Foundations of Computing and Decision Sciences*, 37(3): 223, 2012.

[11] D. E. Goldberg. *Optimization, and machine learning*. Addison-Wesley, 1989.

[12] P. Hansen, N. Mladenović, R. Todosijević, and S. Hanafi. Variable

neighborhood search: basics and variants. *EURO J. Comput. Optim.*, 5 (3):423–454, 2017.

[13] T. C. Hu. Optimum communication spanning trees. *SIAM Journal on Computing*, 3(3):188–195, 1974.

[14] D. S. Johnson, J. K. Lenstra, and A. R. Kan. The complexity of the network design problem. *Networks*, 8(4):279–285, 1978.

[15] M. Juvan, J. Marincek, and B. Mohar. Embedding a graph into the torus in linear time. In *Lecture Notes in Computer Science*, pages 360–363. Springer, Berlin/Heidelberg, 1995.

[16] A.-M. Kermarrec and C. Thraves. *Can everybody sit closer to their friends than their enemies?*, page 388–399. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[17] M. Klugerman, A. Russell, and R. Sundaram. On embedding complete graphs into hypercubes. *Discrete Mathematics*, 186(1–3):289–293, 1998.

[18] Y.-X. Lin. Two-dimensional bandwidth problem. In *Combinatorics, Graph Theory, Algorithms and Applications 1993*, pages 223–232, Beijing, 1994.

[19] M. Livingston and Q. F. Stout. Embeddings in hypercubes. *Mathematical and Computer Modelling*, 11(0):222–227, 1988.

[20] M. Lozano, A. Duarte, F. Gortázar, and R. Martí. A hybrid metaheuristic for the cyclic antibandwidth problem. *Knowledge-Based Systems*, 54:103–113, 2013.

[21] X. Ma, J. Wu, S. Xue, J. Yang, C. Zhou, Q. Z. Sheng, H. Xiong, and L. Akoglu. A comprehensive survey on graph anomaly detection with deep learning. *IEEE transactions on knowledge and data engineering*, 35(12):12012–12038, 2021.

[22] Z. Miller and I. H. Sudborough. A polynomial algorithm for recognizing bounded cutwidth in hypergraphs. *Mathematical systems theory*, 24:11–40, 1991.

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

[24] E. G. Pardo, N. Mladenović, J. J. Pantrigo, and A. Duarte. Variable formulation search for the cutwidth minimization problem. *Applied Soft Computing*, 13(5):2242–2252, 2013.

[25] E. G. Pardo, M. Soto, and C. T. Caro. Embedding signed graphs in the line. *Journal of Combinatorial Optimization*, 29(2):451–471, 2015.

[26] E. G. Pardo, R. Martí, and A. Duarte. Linear layout problems. In R. Martí, P. M. Pardalos, and M. G. C. Resende, editors, *Handbook of Heuristics*, pages 1025–1049. Springer, 2018.

[27] E. G. Pardo, A. García-Sánchez, M. Sevaux, and A. Duarte. Basic variable neighborhood search for the minimum sitting arrangement problem. *Journal of heuristics*, (2):249–268, 2020.

[28] A. Raspaud, H. Schröder, O. Sýkora, L. Torok, and I. Vrt'o. Antibandwidth and cyclic antibandwidth of meshes and hypercubes. *Discrete Mathematics*, 309(11):3541–3552, 2009.

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

[30] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[31] L. van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[32] J. R. Woodcock. *A faster algorithm for torus embedding.* PhD thesis, University of Victoria, 2006. PhD Thesis.

[33] C. Álvarez, R. Cases, J. Díaz, J. Petit, and M. Serna. Approximation and randomized algorithms in communication networks. in: Routing trees problems on random graphs. In *ICALP Workshops 2000*, pages 99–111. Carleton Scientific, 2000.