UNIVERSIDAD DE CARABOBO

Facultad Experimental de Ciencias y Tecnología

Licenciatura en Computación

OPTIMIZACIÓN BASADA EN ALGORITMOS GENÉTICOS PARA LA RESOLUCIÓN DEL PROBLEMA DEL ÍNDICE DE TRANSMISIÓN SOBRE ENLACES DE UN GRAFO

Elaborado por:

Ing. Victor Cesar Mendoza Alarcon

V- 21.476.548

Tutor: **Dr. Amadís Antonio Martínez-Morales**

Noviembre de 2019

Agradecimientos:

En primer lugar quiero agradecer a mis padres Selena Alarcon y Victor Mendoza por todo, especialmente a mi madre que estuvo conmigo en todo este proceso apoyándome, soportándome y mucho mas importante, por creer en mi.

Seguidamente quiero agradecer a mi hermano, Miguel, mis abuelos Olinda y Cesar, mis tíos Karen y Juan, y a mis primos Sebastian y Valentina, por todo el apoyo, la ayuda brindada y recordarme que desistir no es una opción cuando uno quiere algo.

Quiero agradecer a mi tutor Amadis Martinez por toda la ayuda brindada en este trabajo.

Quiero agradecer también a Mayerling Aguilera, amiga que conocí en el transcurso de la carrera y que fue primordial en esta ultima etapa, siempre estuvo apoyándome y dándome alegría para poder seguir, una gran persona que no olvidare.

Un agradecimiento muy especial a Marinés González, que a pesar de la distancia siempre tomaba un poco de su tiempo para apoyarme y darme ánimos cuando estos ya no estaban, alguien que me hizo entender cosas muy cruciales que eran necesarias para poder continuar y lograr lo presentado en esta investigación.

Quiero agradecer también a mis compañēros Alvaro Espinoza y Jesus Valdez, con los cuales estuve trabajando compartiendo información para que cada uno pudiéramos concluir nuestro Trabajo Especial de Grado, también se agradece a Jesus Valdez por facilitarme los servidores patrocinados por Bita Indexes para los experimentos.

Por ultimo, quiero agradecer a todos los compañeros que estuvieron conmigo desde el comienzo de la carrera, que siempre estuvieron apoyándome y creyendo en mi, se les aprecia.

UNIVERSIDAD DE CARABOBO

Facultad Experimental de Ciencias y Tecnología

Licenciatura en Computación

OPTIMIZACIÓN BASADA EN ALGORITMOS GENÉTICOS PARA LA RESOLUCIÓN DEL PROBLEMA DEL ÍNDICE DE TRANSMISIÓN SOBRE ENLACES DE UN GRAFO

Elaborado por:

Ing. Victor Cesar Mendoza Alarcon

V- 21.476.548

Tutor Académico:

Dr. Amadís Antonio Martínez-Morales

RESUMEN

Uno de los mayores retos al momento de diseñar una arquitectura de red es la de definir una configuración inicial para las conexiones de los distintos nodos de forma tal que no sature las interconexiones entre un par de nodos, a esta capacidad de no saturar las conexiones entre nodos se le conoce como índice de transmisión sobre enlaces. Dado que una red Γ puede ser representada como un grafo G no dirigido de orden n y una tabla de enrutamiento R con $n(n-1)$ caminos entre par de nodos (u,v), se define la congestión de un enlace e en un enrutamiento R sobre G como el número de caminos en R que pasan a través de e, y se denota por $\pi(G,R,e)$. A su vez, el índice de transmisión de enlace de la red $\pi(G,R)$ es el número máximo de caminos de R que pasan a través de cualquier enlace de G. El índice de transmisión sobre enlaces de G, denotado por $\pi(G)$, se define como el número mínimo de $\pi(G,R)$ entre todos los posibles enrutamientos R sobre G. El problema del cálculo del índice de transmisión sobre enlaces de un grafo es NP-Completo, y es considerado de gran interés a nivel teórico y práctico. Por este motivo, en este trabajo especial de grado se propone una solución, basada en algoritmos genéticos, para el cálculo del índice de transmisión sobre enlaces de un grafo.

Palabras clave: Algoritmos genéticos, índice de transmisión sobre enlaces, grafos, enrutamiento, redes de comunicaciones, metaheurísticas.

UNIVERSIDAD DE CARABOBO

Facultad Experimental de Ciencias y Tecnología

Licenciatura en Computación

OPTIMIZACIÓN BASADA EN ALGORITMOS GENÉTICOS PARA LA RESOLUCIÓN DEL PROBLEMA DEL ÍNDICE DE TRANSMISIÓN SOBRE ENLACES DE UN GRAFO

Elaborado por:

Ing. Victor Cesar Mendoza Alarcon

V- 21.476.548

Tutor Académico:

Dr. Amadís Antonio Martínez-Morales

ABSTRACT

One of the biggest challenges when designing a network architecture is defining an initial configuration for the different node connections in such way that it won't saturate the interconnections between a pair of nodes, this capacity of not saturating the connection amongst nodes is known as edge forwarding index. Since a Γ network can be represented as an G non-directed graph of order n and a R routing table with $n(n-1)$ paths between pair of nodes (u,v), the congestion of a e link in a R routing over G is defined as the number of paths in R that pass through e, and is denoted by $\pi(G,R,e)$. In turn, the edge forwarding index $\pi(G,R)$ is the maximum number of paths in R that pass through any edge in G. The edge forwarding index on G, denoted by $\pi(G)$, is defined as the minimum number of $\pi(G,R)$ among all possible routing R over G. The problem of calculating the edge forwarding index in a graph is NP-Complete, and is considered of great interest at a theoretical and practical level. For this reason, in this special grade work a solution is proposed, based on genetic algorithms, for the calculation of the edge forwarding index of a graph.

Keywords: Genetic algorithms, edge forwarding index, graphs, routing, communications networks, metaheuristics.

Índice

Índice de Figuras

Índice de cuadros

Índice de algoritmos

Introducción

Una red de computadoras o de comunicaciones es un sistema de transmisión de información a través de distintos dispositivos periféricos interconectados. Estos dispositivos son denominados nodos, y las interconexiones entre ellos se denominan enlaces (medio por el cual la información es transmitida). Para que la red pueda entregar correctamente el mensaje (información) desde un nodo origen hacia un nodo destino, se debe planificar una ruta óptima desde el origen hacia el destino, sin saturar los enlaces o nodos que se encuentren en el camino.

Cuando en una red todos los dispositivos se encuentran conectados entre sí, la construcción de un enrutamiento no suele ser un problema. En este caso, la complejidad de encontrar las mejores rutas en una red sin que ocurra una sobrecarga pasa a ser más sencillo, ya que existe un enlace entre cada par de nodos. Sin embargo, en las redes donde existen dispositivos que no se encuentran conectados directamente, el problema de hallar las rutas suele ser de una mayor complejidad, ya que que el diseñador deberá especificar un camino entre cada par de nodos en la red, sin sobrecargar ya sea los enlaces o los nodos. Este conjunto de caminos se denomina enrutamiento.

Formalmente, se define una red $\Gamma(G,R)$ como un grafo no dirigido $G(V,E)$ de orden n, donde n indica el número de vértices (nodos) en V y un conjunto de enlaces E. Por otro lado, se tiene un conjunto C de $n(n-1)$ caminos en un enrutamiento $R(V,C)$, donde un camino $c_i \in C$ representa una ruta para cada par ordenado de vértices $(u,v) \in V$. Al definir el problema utilizando esta notación se observa que, dado un enrutamiento R, existen diversos caminos que pasan a través de algún nodo o enlace, implicando que éstos puedan sobrecargarse, afectando la funcionalidad y correcta transmisión en la red.

Este trabajo especial de grado se enfocará en el problema relacionado a la sobrecarga de los enlaces en la red. Se tiene que un enrutamiento R debe poseer un conjunto de caminos (u,v) entre un nodo inicial u y un nodo final v de forma que el enlace $e \in E$ no se sobrecargue, por lo que si se quiere buscar un enrutamiento R para una red Γ en un grafo G se define el índice de transmisión de enlaces [1] como métrica para la medición de la congestión en un enlace e dentro de una red Γ en términos del número de caminos que pasan por cada enlace $e \in E$.

El problema del índice de transmisión sobre un enlace fue motivado al querer maximizar la capacidad en una red, por lo que si se quiere aprovechar al máximo el potencial de una red se debe definir un enrutamiento R sobre el grafo asociado G tal que minimice la carga de cada enlace. El cálculo de este índice de transmisión en un grafo es un problema NP-Completo [2], por lo que ganó relevancia en el área de la optimización combinatoria, originando así un gran interés en el uso de metaheurísticas para conseguir una solución eficiente en un tiempo computacional razonable.

La metáfora de la teoría de la evolución [3] dio como origen en la clase de los algoritmos basados en población de las metaheurísticas a los algoritmos genéticos. Este tipo de algoritmos generan una población P (conjunto) de μ individuos p_i (soluciones) donde cada individuo p_i es evaluado a través de una función de idoneidad [4] (función de costo o aptitud) Ω que permite determinar la aptitud de cada p_i, se aplica una función de selección ψ que permita evaluar y seleccionar un par de individuos (p_i, p_j) como los candidatos a generar la siguiente población P^* mediante una función de cruce ϕ. Cada nuevo individuo p_i^* en la población P^* de la nueva generación esta sujeta a una probabilidad α de mutación para así aumentar la diversidad de las soluciones en la población.

En el proyecto presentado se hará uso de los algoritmos genéticos para hallar un enrutamiento R que sea solución factible para el problema del índice de transmisión de enlace. El presente documento se divide en cuatro secciones sin tomar en cuenta la introducción. En la sección 1 se engloba la motivación del estudio, el planteamiento del problema, justificación y objetivos de la investigación. La sección 2 presenta los antecedentes y bases teóricas. La metodología a utilizar se detalla en la sección 3. El desarrollo del algoritmo se encuentra en la sección 4. En la sección 5 se detallaran los experimentos realizados al algoritmo genético para distintas topologías de grafos. Por ultimo, en la sección 6 se concluirán sobre los experimentos realizados junto a las recomendaciones y trabajos futuros.

1. El Problema

1.1. Motivación

En las ciencias computacionales existen una gran cantidad de problemas clasificados de acuerdo a su complejidad. Muchos de estos problemas se pueden clasificar en la categoría P (problemas con solución en tiempo polinomial $O(n^k)$ con k como una constante y n como el número de entradas), por otro lado se encuentran los problemas con complejidad NP (problemas no deterministas en tiempo polinomial $O(2^{cn})$ donde c es una constante mayor que cero [5]).

Los problemas del tipo NP-Completo es un subconjunto de los problemas NP y pueden describirse de manera informal como los más difíciles dentro del conjunto NP [6]. Al encontrar un algoritmo de tiempo polinómico para al menos uno de los problemas entonces existirá un algoritmo polinómico para todos los problemas en NP. Formalmente podemos definir un problema NP-Completo como un problema de decisión C contenido en NP y todo problema de NP es reducible a C en tiempo polinomial.

Esto ha llevado a que exista un gran interés es buscar soluciones de tiempo polinomial que den solución a problemas NP-Completo, pero sin ningún resultado concluyente en la actualidad, las demostraciones de los problemas NP-Completo infieren en la posibilidad de que exista un algoritmo polinomial que le de solución. El problema a tratar se considera del tipo NP-Completo [2] y entra en la categoría de problemas sobre grafos, dado que buscar una solución polinomial para este tipo de problemas no es fácil se suele buscar una solución óptima o razonable que satisfaga el problema pero no la mejor. Los problemas NP-Completo son de sumo interés en la computación teórica por lo que se aportará una solución razonable al problema del índice de transmisión de enlace haciendo uso de algoritmos genéticos.

1.2. Planteamiento del Problema

Sea $G = (V, E)$ un grafo no dirigido simple y conexo de orden n, donde $V = V(G)$ es el conjunto de vértices de G y $E = E(G)$ es el conjunto de enlaces de G, por otro lado se define el orden de G como $n = |V|$ y $d = |E|$ como el tamaño de G.

Dado un par ordenado de vértices $(u, v) \in V : u \neq v$ se define un camino c en G desde u hasta v como una secuencia de vértices $c(u, v) = \{u = v_0, v_1, ..., v_k = v\} : \langle \forall_i : 1 \leq i \leq k : \{v_{i-1}, v_i\} \in E \rangle$. Se define $u = v_0$ y $v = v_k$ como extremos de c y $\{v_1, v_2...., v_{k-1}\}$ como vértices interiores de c. Si todos los vértices $v \in c$ son distintos entre si, se dice que c es un camino elemental entre u y v.

Un enrutamiento $R = R(G)$ en G es un conjunto de caminos elementales que conectan cada par de vértices (u, v) distintos en G.

$$R = \{c(u, v) : (u, v \in V) \wedge (u \neq v)\}$$

En [7] se dice que un enrutamiento R en G es mínimo R_m si $\forall c \in R$ es el más corto; R es simétrico o bidireccional si $\forall c \in R\ c_i(u, v) = c_i(v, u) : \{(u, v) \in V \wedge u \neq v\}$ y se representa con R_s; se dice que R es consistente si para un par de vértices (u, v) distintos y para cada vértice interior z del camino $c(u, v) \in R$, el camino $c(u, v)$ es la concatenación de los caminos $c(u, z) \wedge c(z, v)$.

Una red $\Gamma = \Gamma(G, R)$ de define como un par (G, R) donde G es un grafo no dirigido, simple y conexo, y R un enrutamiento sobre G. En una red Γ pueden existir muchos caminos que pasen a través de un enlace $e \in E$, lo cual significa que el enrutamiento R sobrecarga el enlace e. De manera intuitiva un "buen" enrutamiento R es aquel que no sobrecargue ningún enlace e, es decir que no muchos caminos $c \in R$ pasen a través de ellos.

Para medir la congestión en un enlace e se utiliza el concepto de índice de transmisión de enlace (*edge-forwarding index*) introducida en [1].

Sea $\Re(G)$ el conjunto de todos los enrutamientos R sobre G, dado un enrutamiento $R \in \Re(G)$ y un enlace $e \in E$ se define la congestión del enlace e como el número de caminos $c_i \in R$ que pasan a través de e y es denotado por $\pi(G,R,e) = \pi(\Gamma,e) : e \in E \wedge E \in \Gamma$.

El índice de transmisión sobre enlaces en una red Γ, denotada por $\pi(G,R) = \pi(\Gamma)$ es el número máximo de caminos en R que pasan a través de cualquier enlace $e \in E$.

$$\pi(\Gamma) = max_{e \in E}\{\pi(\Gamma,e) : E \in \Gamma\}$$

Análogamente, el índice de transmisión de enlace en un grafo G es denotado por $\pi(G)$ y se define como el número mínimo de los índices de transmisión sobre enlaces $e \in E$ de una red Γ, considerando todos los enrutamientos R posibles en $\Re(G)$ sobre el grafo G.

$$\pi(G) = min_{R \in \Re(G)}\{\pi(\Gamma) : R \in \Gamma\}$$

A continuación se ilustrara lo dicho anteriormente a través de un ejemplo haciendo uso del grafo G presentado en la **Fig. 1**, tomando como todo el conjunto de enrutamientos posibles $\Re = \{R_0, R_1\}$ y un conjunto de enlaces E, se calculara el índice de transmisión sobre enlaces de un grafo G.

$$\Re = \left\{ R_0 = \begin{bmatrix} \emptyset & \{1,2\} & \{1,4,3\} & \{1,2,4\} \\ \{2,1\} & \emptyset & \{2,4,3\} & \{2,4\} \\ \{3,1\} & \{3,2\} & \emptyset & \{3,4\} \\ \{4,2,1\} & \{4,3,2\} & \{4,3\} & \emptyset \end{bmatrix}, R_1 = \begin{bmatrix} \emptyset & \{1,2\} & \{1,3\} & \{1,4\} \\ \{2,1\} & \emptyset & \{2,3\} & \{2,4\} \\ \{3,1\} & \{3,2\} & \emptyset & \{3,4\} \\ \{4,1\} & \{4,2\} & \{4,3\} & \emptyset \end{bmatrix} \right\}$$

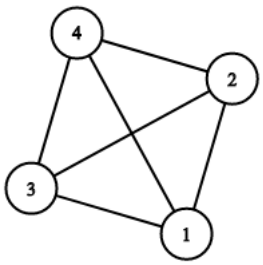

Figura 1: Grafo de orden n=4

$$E = \{(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)\}$$

Para el calculo del índice de transmisión sobre enlaces de un grafo $\pi(G)$, se requiere conocer la mínima congestión de una red definida como $\pi(\Gamma)$. Se define el conjunto $\Gamma = \{(G,R_0),(G,R_1)\}$ como todas las posibles redes Γ de un grafo G dado un enrutamiento $R \in \Re$. La congestión de una red se define como la máxima congestión (o cantidad de caminos que atraviesan) en un enlace $e \in E$ dentro de un red Γ y es denota por $\pi(\Gamma, e)$.

Al momento de conocer la congestión de todos los enlaces de un conjunto de redes Γ tenemos:

$$\{\pi(G,R_0,e),\pi(G,R_1,e)\} : e \in E$$

Para el conjunto de enrutamientos $\{R_0,R_1\}$ la congestión de todos los enlaces se define como:

$$\pi(G,R_0,e) = \{4,1,1,2,4,5\}$$

$$\pi(G,R_1,e) = \{2,2,2,2,2,2\}$$

Conociendo la congestión de cada enlace dentro de una red se procede a obtener la máxima carga de todos los enlaces e dentro del conjunto Γ:

$$\pi(\Gamma) = max_{e\in E}\{\pi(G,R_0,e),\pi(G,R_1,e)\}$$

$$\pi(\Gamma) = max\{\{4,1,1,2,4,5\}_{R_0},\{2,2,2,2,2,2\}_{R_1}\}$$

$$\pi(\Gamma) = \{5_{R_0},2_{R_1}\}$$

Una vez obtenido los índices de transmisión sobre todos los enlaces e para todas las posibles redes $(G,R) : R \in \Re$ se define el índice de transmisión sobre enlaces de un grafo como:

$$\pi(G) = min_{R\in\Re}\pi(\Gamma)$$

$$\pi(G) = min\{5_{R_0},2_{R_1}\}$$

$$\pi(G) = 2_{R_1}$$

Siendo así el enrutamiento R_1 de todos los posibles en $\Re$ aquel que donde el índice de transmisión sobre enlaces de un grafo es mínimo.

Por lo que podemos concluir que dado un grafo G de orden n, el problema del índice de transmisión de enlaces consiste en encontrar un enrutamiento $R \in \Re(G) : min_R\pi(G)$, por lo que para hallar una solución satisfactoria al problema en un tiempo computacional razonable, se propondrá una implementación haciendo uso de algoritmos genéticos.

1.3. Justificación

El problema del índice de transmisión fue motivado por la maximización de la capacidad de una red de comunicación y fue propuesto en [8] y fue definido para los vértices o nodos de la red, posteriormente en [1] fue presentado el problema del índice de transmisión pero haciendo referencia a los enlaces o conexiones de la red.

De esta manera, el problema de la maximización de la capacidad de la red se reduce a la minimización del índice de transmisión sobre los enlaces de un enrutamiento sobre el grafo asociado. Si la capacidad de una red de comunicaciones puede ser aprovechada por completo o no, dependerá de la selección del enrutamiento sobre el grafo adyacente.

Cuando el problema del índice de transmisión en un grafo fue demostrado como NP-Completo [2] gano el foco de interés tanto a nivel práctico y teórico; a pesar de que el problema ha sido estudiando a nivel teórico en [1, 7, 9] et. al. no se ha encontrado evidencia de una solución computacional.

La falta de una solución computacional ha llevado a la motivación de la utilización de metaheurísticas para resolver el problema, por lo que la relevancia teórica y practica de los índices de transmisión sobre enlaces justifica el desarrollo de un algoritmo basado en metaheurísticas para el calculo de transmisión sobre enlaces de un grafo, haciendo uso de algoritmos genéticos.

1.4. Objetivos Generales y Específicos

1. Desarrollar un algoritmo de optimización combinatoria, basado en la metaheurística de algoritmos genéticos, para el cálculo del índice de transmisión sobre enlaces de un grafo.

   a) Diseñar un algoritmo, basado en algoritmos genéticos, para el cálculo del

índice de transmisión sobre enlaces de un grafo.

b) Implementar el algoritmo previamente diseñado, utilizando un lenguaje de programación de alto nivel, para el cálculo del índice de transmisión sobre enlaces de un grafo.

2. Realizar estudios empíricos, utilizando métricas adecuadas, para la evaluación de desempeño del algoritmo implementado.

   a) Desarrollar un estudio experimental, utilizando métricas adecuadas, datos sintetizados y entonación de parámetros, para la evaluación de desempeño del algoritmo desarrollado.

   b) Analizar los resultados obtenidos por el algoritmo implementado, para la deducción de conclusiones.

2. Marco Teórico

2.1. Antecedentes

Para la elaboración de esta investigación, se hizo necesario revisar estudios preliminares que estuviesen relacionados con el tema a tratar. Un antecedente, es todo aquel trabajo que precede a la investigación que se esté realizando y contribuyan en cierta medida con la misma. Dado que el presente proyecto no posee una solución computacional en la actualidad, se hablará del problema en su formalidad matemática.

El problema original del índice de transmisión fue planteado por Fan Chung, E. Coffman, M. Reiman, y B. Simon [8], los cuales definieron el índice de transmisión de una red como el número de caminos que pasa por cualquier vértice en un grafo. El trabajo consistió en el calculo de los índices para las redes cubo y se generalizaron para las redes *Bruijn*. Finalmente, presentaron técnicas eficientes para construir grandes redes con pequeños índices de transmisión.

En [1] los autores M.C. Heydemann, J.C. Meyer y D. Sotteau definieron formalmente el problema del índice de transmisión de enlace como una red a través de un grafo no dirigido G y un conjunto R de $n(n-1)$ caminos entre todos los vértices. Así mismo definieron $\pi(G,R,e)$ como la cantidad de caminos de R que pasan a través de e, a su vez $\pi(G,R)$ indica la congestión de un enlace de la red como el máximo número de caminos de R que pasan a través de cualquier enlace de G. Por último el índice de transmisión de un grafo G se define como el mínimo $\pi(G,R)$ sobre todos los enrutamientos de R en G y se denota como $\pi(G)$.

Abdelmadjid Bouabdallah y Dominique Sotteau definen en [9] el índice de transmisión de enlace para grafos pequeños. Los autores presentan resultados para un grafo de orden $n < 15$ y un grado $\Delta \leq n-1$ determinaron $\pi_{\Delta,n}$ como el mínimo

$\pi(G)$ sobre todos los grafos G de orden n con un grado máximo de al menos Δ. Los resultados no lograron encontrar una solución general para cualquier n si Δ es lo suficientemente grande.

2.2. Bases Teóricas

2.2.1. Estructura de Datos

El propósito primario de los programas de computadoras no es la de realizar cálculos, si no, en el almacenamiento y recuperación de información lo mas rápido posible, por lo que la forma en que la información se representa es fundamental en las ciencias de la computación. Por esta razón el estudio de las estructuras de datos junto al uso de algoritmos para manipularla siendo así el corazón de la ciencia de la computación.

En un sentido general, una estructura de datos es una representación de cualquier tipo de información que se encuentra asociada a un conjunto de operaciones. Comúnmente el término estructura de datos significa una organización o estructura para una colección de datos.

2.2.2. Algoritmos

Un algoritmo se puede definir como una herramienta para resolver un problema computacional bien especificado. para lograr esto se definen procedimientos que toman algún valor, o conjunto de valores, como entrada y son procesados hacia alguna valor, o conjunto de valores, conocidos como salida. Por lo que un algoritmo es una secuencia de pasos computacionales para procesar una entrada en una salida.

2.2.3. Teoría de Grafos

En ciencias de la computación la teoría de grafos es el estudio de una estructura matemática utilizada para modelar relaciones o procesos dinámicos entres distin-

tos objetos de una determinada colección. Informalmente un grafo es un conjunto de objetos llamados vértices (o nodos) conectados por enlaces (o arcos). Dependiendo de la aplicación, los enlaces pueden poseer dirección o no; pueden unir un vértice a si mismo si es permitido; se les puede asignar un peso. La notación utilizada para representar los grafos es la utilizada en las matemáticas.

Un grafo puede ser representado en un sistema computacional de cuatro distintas formas: lista de incidencia, matriz de incidencia, lista de adyacencia y matriz de adyacencia. Usualmente las representaciones de incidencias son útiles cuando la información sobre los enlaces es más deseable que la de los vértices, en cambio la representación de adyacencia es el caso contrario y suele ser la representación más utilizada dado que en las aplicaciones la información en los vértices es más deseada.

2.2.4. Metaheurísticas

La metaheurística es un marco de trabajo [10] para la resolución de problemas de optimización con una complejidad combinatoria siguiendo un esquema algorítmico bajo determinados lineamientos o guías para la construcción de algoritmos heurísticos de optimización; el termino también suele referirse a una implementación específica [10] del marco de trabajo. Los algoritmos heurísticos conforman una categoría en los problemas de optimización en los cuales no se garantiza que la solución obtenida sea la óptima por lo que usualmente suele devolver resultados por debajo del óptimo. El marco de trabajo se basa en explorar todo el conjunto de soluciones de un problema e ir encontrando la mejor solución posible.

2.2.5. Algoritmos Genéticos

Los algoritmos genéticos son un marco de trabajo dentro de las metaheurísticas basadas en poblaciones los cuales permiten realizar búsquedas basadas en los principios de la selección natural y genética; fue introducido por J. Holland en 1970.

El espacio del problema se abstrae como una población de individuos y trata de explorar los individuos mas aptos para producir la siguiente población, de esta manera el algoritmo puede resolver de manera óptima el problema.

El esquema de trabajo del algoritmo genético consiste en un conjunto de pasos de forma tal que permita ir evolucionando las soluciones hasta hallar una solución óptima, los pasos se pueden definir como:

1. **Población inicial:** Se genera una población de μ soluciones (individuos) posibles para el problema planteado a través de una función Ξ.
2. **Selección de individuos:** Tomando en cuenta una función Ω denominadas función de costo, se selecciona el mejor par de individuos tal que $min(\Omega)$ (conocidos también como padres) a través de una función de selección Ψ.
3. **Generación de individuos:** Con los mejores individuos obtenidos en el paso 3, se generan nuevos individuos mediante una función de cruce Φ.
4. **Mutación:** Cada nuevo individuo generado posee una probabilidad α de mutación para así permitir variaciones, se denota con la función Υ.
5. **Nueva población:** Se construye una nueva población tomando en cuenta los individuos seleccionados y los generados, se representa con una función Θ.
6. **Evolución:** Se repiten los pasos del 2-6 hasta que se cumpla la condición de terminación, la cual generalmente suele ser una cantidad β de generaciones.
7. **Solución:** Una vez concluida las mejores se selecciona el mejor individuo en la población con una función η.

3. Marco Metodológico

En el marco metodológico se definirán los pasos y procedimientos que habría que seguir con el fin de garantizar y cumplir con los objetivos establecidos en cada etapa de la investigación. El proyecto está enmarcado en la investigación de tipo experimental en combinación con la metodología *DIMMA*.

3.1. Tipo de Investigación

La investigación experimental es un proceso que consiste en someter a un objeto o grupo de individuos en determinadas condiciones, estímulos o tratamiento (variable independiente), para observar los efectos o reacciones que se producen (variable dependiente) [11].

3.2. DIMMA

En [12] los autores Yaghini, Masoud y Akhavan, Rahim proponen en 2010 la metodología de diseño e implementación para Algoritmos metaheurísticos (*Design and Implementation Methodology for Metaheuristic Algorithms - DIMMA*), donde se definen directrices para todos los que quieran diseñar e implementar un algoritmo metaheurístico.

DIMMA tiene dos dimensiones que incluyen dimensión dinámica y de disciplina. La dimensión dinámica es la dimensión horizontal, que incluye fases de la metodología: iniciación, *blueprint* y construcción. La dimensión de la disciplina es la dimensión vertical que muestra las disciplinas, las cuales se agrupan lógicamente los pasos, actividades y artefactos [12].

Las fases metodológicas del *DIMMA* poseen un conjunto de pasos a seguir, donde cada paso puede reiterarse de manera que en cada iteración se consiga alguna mejora por lo que no están limitados a seguir las fases y esquemas de forma com-

pletamente lineal, los pasos a seguir se pueden definir como:

- **Iniciación.**
  - Antecedentes y definición del problema.
  - Definición de objetivos.
  - Selecciona de instancias para la evaluación del algoritmo.
- ***Blueprint.***
  - Método de solución metaheurística.
  - Definir medidas de rendimiento.
  - Seleccionar estructura de datos.
  - Diseño del algoritmo.
- **Construcción.**
  - Implementación del algoritmo.
  - Ajuste de parámetros.
  - Análisis de rendimiento del algoritmo.
  - Documentación de resultados.

4. Diseño de la Solución

4.1. Estructuras de Datos

4.1.1. Grafo

La representación de un grafo como estructura vendrá dada por una lista de adyacencia con la variación de que las listas estarán basadas en mapas(tabla de *hash*). La selección de representar tanto los vértices y la lista de adyacencia a través de un mapa viene dada por la facilidad que trae esta representación al momento de consultar sus elementos.

Al analizar la estructura seleccionada haciendo uso de la notación O grande, la complejidad en espacio para almacenar todos los vértices la cual sera representada como un entero cuya complejidad es $O(1)$, se define el conjunto V como todos los vértices asociadas a un grafo G como:

$$V = \{v_1, v_2, \ldots, v_d\}$$

Donde d denota el numero de vértices $v \in V$, se define $|V| = d$ es la cantidad de elementos en V; dado que cada vértice v representa un entero y teniendo $|V|$ elementos en V la complejidad en espacio para almacenar todos los vértices viene dada por la cantidad de los mismos y se denota como $O(|V|)$.

Sea $v_i \in V$ el i-ésimo vértice se define el mapa de adyacencia de todos los vértices $A_i = \{v_{ik}\} : 1 \leq i \leq d \wedge 1 \leq k \leq d-1 \wedge i \neq k$ tal que v_{ik} es adyacente al vértice v_i y $|A_i| = k$ como la cantidad de elementos en A_i. Cada v_{ik} viene representado por un entero con complejidad $O(1)$, dado todos los vértices v_i la complejidad de almacenar todos los vértices adyacentes A_i viene denotado por $|E|$ y su complejidad es:

$$O\left(\sum_{i=1}^{|V|}|A_i|\right)$$
$$O(|E|)$$

La complejidad en espacio de almacenar un grafo G a través de un mapa de vértices y mapa de adyacencia es de:

$$O(|V|+|E|)$$

4.1.2. Camino

Un camino $c = \{v_1, \ldots, v_k\} : 2 \leq k \leq |V|$ consta de un camino de vértices desde el vértice v_1 hasta el vértice v_k y sera representado haciendo uso de la estructura de datos tipo arreglo; se define $|c| = k$ como la longitud del camino. Se tiene que un camino c es elemental y simple de manera tal que no existen vértices repetidos.

Cada camino esta representado por un entero con una complejidad en espacio $O(1)$, se dice que la complejidad de almacenar un camino c se encuentra entre:

$$O(2) \leq O(|c|) \leq O(|V|)$$

En el peor de los casos un camino $|c|$ requiere pasar por todos los vértices para llegar de v_1 a v_k, por lo que la complejidad viene dada por $O(|V|)$.

4.1.3. Enrutamiento

Un enrutamiento R consiste en un conjunto de caminos c para todo par de vértices $(i, j) : 1 \leq u, v \leq |V|$, la cantidad de caminos en un enrutamiento viene dada por $|V|^2$.

$$R = \begin{bmatrix} c_{i,j} & c_{i,j+1} & \cdots & c_{i,|V|} \\ c_{i+1,j} & c_{i+1,j+1} & \cdots & c_{i+1,|V|} \\ \vdots & \vdots & \ddots & \vdots \\ c_{|V|,j} & c_{|V|,j+1} & \cdots & c_{|V|,|V|} \end{bmatrix}$$

Cada camino $c_{i,j} : i = j$ representan un conjunto vació por lo que no existe un camino que vaya del vértice v_i, v_i, la cantidad de caminos tal que $i == j$ representan la diagonal de la matriz R, por lo que se puede extraer la diagonal y tenemos que la cantidad de caminos en un enrutamiento viene dado por $|V|\big(|V|-1\big)$ y se utiliza la estructura del tipo arreglo para representarlo. Se define un enrutamiento como:

$$R = \{c^k_{i,j} : \forall i \forall j i \neq j \wedge 1 \leq i, j \leq |V| \wedge 1 \leq k \leq |V|\big(|V|-1\big)\}$$

Cada $c^k_{i,j}$ representa el k-ésimo camino para el par de vértices (i, j) y es representado a través de la estructura Camino presentada anteriormente, por lo que la complejidad en espacio para un camino $c^k_{i,j} = O(|V|)$, esto conlleva a que la complejidad de la estructura Enrutamiento venga denotada por:

$$O\left(\sum_{k=1}^{|V|\left(|V|-1\right)} c^k_{i,j}\right)$$

$$O\left(\sum_{k=1}^{|V|\left(|V|-1\right)} |V|\right)$$

$$O\Big(|V|\big(|V|-1\big)|V|\Big)$$

$$O\Big(|V|^2\big(|V|-1\big)\Big)$$

4.1.4. Población

La población $P = \{p_r : 1 \leq r \leq \mu\}$ representa el conjunto inicial de soluciones la cual viene dado por μ enrutamientos solución, esta sera representada a través

de un arreglo de longitud $|P| = \mu$ donde cada p_r representa una estructura tipo Enrutamiento y puede definirse $p_r = c_{i,j}^{r,k} : 1 \leq i, j \leq |V| \wedge i \neq j \wedge 1 \leq k \leq |V|(|V| - 1) \wedge 1 \leq r \leq \mu$ como el k-ésimo camino del r-ésimo enrutamiento entre los vértices (i, j) con una complejidad de $O\Big(|V|^2\big(|V| - 1\big)\Big)$, por lo tanto, la complejidad en espacio que alberga el conjunto de soluciones P viene dada por:

$$O\left(\sum_{r=1}^{\mu} p_r\right)$$

$$O\Big(\mu|V|^2\big(|V| - 1\big)\Big) : \mu \in \mathbb{N}$$

4.2. Representación de las Estructuras

El algoritmo a presentar en este trabajo hace uso de las estructuras mencionadas anteriormente, para su representación se hará uso de la notación NASPI (**Algoritmo. 2)** el cual es un lenguaje pseudoformal que ofrece: (1) una estructura básica de algoritmos, (2) los tipos de datos elementales y estructurados más comunes, (3) declaraciones de constantes y variables asociadas a cualquier tipo de dato, (4) operaciones de entrada/salida básicas, (5) procedimientos y funciones parametrizadas, (6) los tipos de paso de parámetro básicos, y (7) la definición de tipos de datos por parte del usuario [13].

Dado que en la notación *NASPI* no existe una forma de representar la estructura de tablas de *hash* , se representara el mapeo a través de la estructura **arreglo** de dos dimensiones, pero su comportamiento se tomara como tablas de *hash*. Por otro lado, la estructura camino, enrutamiento y población serán representadas a través de un **arreglo**, en el caso de la longitud del camino viene dada por $1 \leq k \leq |V|$, la del enrutamiento viene siendo $1 \leq r \leq |V|(|V| - 1)$, por ultimo la población sera de longitud $\mu \in \mathbb{N}$.

tipo
Grafo = **arreglo** $[1 \cdots |V|, 1 \cdots |A_i|]$ **de entero**
Camino = **arreglo** $[1 \cdots k]$ **de entero**
Enrutamiento = **arreglo** $[1 \cdots |V|(|V|-1)]$ **de Camino**
Poblacion = **arreglo** $[1 \cdots \mu]$ **de Enrutamiento**
Vertice = **entero**

Algoritmo 1: Tipo de datos definidos

Por otro lado, se definen las funciones y procedimientos con las cuales se trabaja las estructuras indicando la complejidad de cada una (**Algoritmo. 1**).

func numero_vertices(**Grafo** g) : **entero** // Retorna $|V|$: $|V|$ es constante $O(1)$
func retornar_ady(**Grafo g, Vertice** v) : **arreglo** $[1 \cdots |A_i|]$ **de entero** // Retorna la tabla *hash* adyacente al vértice v $O(1)$
func vertice_aleatorio(**Grafo** g) : **Vertice** // Retorna un vértice aleatorio entre $1 \cdots |V|$ $O(1)$
proc insertar_camino(**Camino** c, **Vertice** u) // Inserta un vértice en la ultima posición del camino $O(1)$
func encontrar_camino(**Camino** c, **Vertice** u) : **Booleano** // **verdadero** si $u \in c$, **falso** en caso contrario $O(|V|)$
func longitud(**Camino** c) : **entero** // Retorna $|c|$: $|c|$ es constante $O(1)$
proc insertar_enrutamiento(**Enrutamiento** r, **Camino** c) // Inserta un camino c en la ultima posición del enrutamiento $O(1)$
proc insertar_poblacion(**Poblacion** p, **Enrutamiento** r) // Inserta un enrutamiento r en la ultima posición de la población $O(1)$
func generar_probabilidad() : **real** // Retorna un valor $p : 0 \leq p \leq 1$ $O(1)$
func generar_camino(**Grafo** g, **Vertice** u, **Vertice** v): **Camino** // Encuentra un Camino desde el vértice u hasta v mediante una variante del BFS $O(|V|+|E|)$
func max(**arreglo** $[1 \cdots m]$ **de entero**) : **entero** // Retorna el mayor elemento en el arreglo Λ $O(m)$
func verificar_enlace(**Camino** c, **Vertice** u, **Vertice** v) : **entero** // Retorna $\pi(\Gamma, e) : e \leftarrow (u, v)$ $O(|V|)$

Algoritmo 2: Funciones y procedimientos definidos

4.3. Diseño del Algoritmo

En la sección 2 se define el esquema general de un algoritmo genético con sus respectivas funciones y operadores. Primeramente se definirán los operadores y funciones del algoritmo genético para luego presentar el esquema en alto nivel junto a su respectivo análisis de complejidad.

4.3.1. Operador de Población Inicial Ξ

Sea $\Re$ el espacio de soluciones de posibles, se define $P \subset \Re$ tal que $p_r = c_{i,j}^{r,k}$: $p_r \in P$ representa el k-ésimo camino del r-ésimo enrutamiento entre todo para de

vértices $(i, j) : i \neq j$. El operador Ξ consiste en generar μ individuos, en este caso viene representado por un $c_{i,j}^{r,k}$ de manera que para cada $r : 1 \leq r \leq \mu$ represente un enrutamiento, el **Algoritmo 3** viene dado por los pasos:

1. Se inicializa el conjunto $P = \{\}$.
2. Se genera un enrutamiento valido dado un grafo G.
3. Se almacena el enrutamiento generado en P
4. Se repiten los pasos del 1 al 3 hasta generar μ enrutamientos.

```
func Ξ(val g: Grafo, var μ: entero) : Poblacion                          
    var:                                                                 
        p : Poblacion                                                    
        r : Enrutamiento                                                 
        i : entero                                                       
    inicio                                                               
        para i ← 1 hasta μ en 1 hacer                          O(μ)      
            r ← generar_enrutamiento(g)                        O(|V|^4)  
            insertar_enrutamiento(p, r)                        O(1)      
        fpara                                                            
        retornar(p)                                            O(1)      
ffunc
```

Algoritmo 3: Algoritmo del operador población inicial Ξ

El operador Ξ consiste en generar una población de enrutamientos de μ individuos donde cada enrutamiento posee una complejidad de $O(|V|^4)$, por lo que la complejidad del **Algoritmo 3** viene dada por $O(\mu(|V|^4))$.

La función **generar_enrutamiento** (**Algoritmo. 4**) consiste en generar $|V|(|V|-1)$ caminos aleatorios entre todo par de vértices $(u, v) : 1 \leq u, v \leq |V| \wedge i \neq j$ y puede definirse con los siguientes pasos:

1. Se inicializa el vértice $u = 1$.
2. Se obtiene la cantidad de $|V|$ de un grafo G asociado.
3. Para un par de vértices $(u, v) : u \neq v$ se genera camino aleatorio.
4. Se almacena el camino generado en el enrutamiento.

5. Si $(v+1)\,\%|V| == 0$ se incremente el vértice $u = u+1$.

6. Se incremente el vértice $v = v+1$.

7. Se repite el proceso desde el paso 3 hasta que $v = |V| * |V|$.

```
func generar_enrutamiento(val g: Grafo) : Enrutamiento                          
    var:                                                                        
        r : Enrutamiento                                                        
        c : Camino                                                              
        u,v : Vertice                                                           
        n : entero                                                              
    inicio                                                                      
        u ← 1                                                          O(1)     
        n ← numero_vertices(g) // n = |V|                              O(1)     
        para v ← 1 hasta n*n en 1 hacer                                O(|V|^2) 
            si (u ≠ v%n) entonces                                      O(1)     
                c ← generar_camino(g,u,v%n)                            O(|V|+|E|)
                insertar_enrutamiento(r, c)                            O(1)     
                si ((v+1)%n == 0) entonces                             O(1)     
                    u ← u+1                                            O(1)     
                fsi                                                             
            fsi                                                                 
        fpara                                                                   
        retonar(r)                                                     O(1)     
ffunc                                                                           
```

Algoritmo 4: Algoritmo para generar un Enrutamiento

La complejidad del **Algoritmo 4** viene dada por el primer ciclo con una complejidad $O(|V|^2)$ y por el generador de camino con una complejidad $O(|V|+|E|)$ para cada iteración del ciclo, resultando así con una complejidad $O(|V|^3+|V|^2|E|)$.

Sabiendo que $|E| = \sum_1^{|V|} |A_i|$ y $|A_i|$ representa la cantidad de elementos adyacentes a un vértice, en el peor de los casos, la lista de adyacencia $|A_i|$ poseerá $|V|-1$ elementos adyacentes por que lo tenemos:

$$|E| = \sum_1^{|V|} |V| - 1$$

$$|E| = |V|(|V|-1)$$

$$|E| = |V|^2 - |V|$$

Al sustituir $|E|$ en la complejidad del algoritmo, tenemos:

$$O(|V|^3 + |V|^2(|V|^2 - |V|))$$
$$O(|V|^3 + |V|^4 - |V|^3)$$
$$O(|V|^4)$$

Con esto obtenemos que la complejidad para el algoritmo que permite generar un enrutamiento viene dado por $O(|V|^4)$ en el peor de los casos.

4.3.2. Función de Costo Ω

Sea $|E|$ el conjunto de todos los enlaces en G, se define $\Lambda_r = \{\lambda_i : \lambda_i^r = \pi(\Gamma, e) \forall e \in E \wedge E \in \Gamma \wedge 1 \leq i \leq |E| \wedge 1 \leq r \leq |V|(|V|-1)\}$ como el vector de congestión para un enlace $e \in E$ del r-ésimo enrutamiento.

Teniendo los enrutamiento $R_1 \rightarrow \Lambda_1 \wedge R_2 \rightarrow \Lambda_2$ asociados a sus respectivos vectores de congestión Λ_r, se dice que el enrutamiento R_1 es mejor que el enrutamiento R_2 si $\pi(\Gamma_1) = max\{\Lambda_1\} : R_1 \in \Gamma_1 < \pi(\Gamma_2) = max\{\Lambda_2\} : R_2 \in \Gamma_2$. Se define la función $\Omega(R_i) \rightarrow max\{\Lambda_i\} : R_i : 1 \leq i \leq |V|(|V|-1)$ para utilizar como métrica de comparación entre enrutamientos R. El **Algoritmo. 5** permite obtener la métrica Ω para un enrutamiento dado a través de los siguientes pasos:

1. Dado un enrutamiento se obtiene su vector de congestión $R \rightarrow \Lambda$.
2. Se retorna el máximo elemento del vector de congestión, siendo este así el costo de un enrutamiento, $max\{\Lambda\}$.

```
func Ω(val r: Enrutamiento, val n: entero : entero
    var:
        Λ : arreglo [1···m] de entero
    inicio:
        Λ ← obtener_lambda(r,n)                          O(|V|^5)
        retornar(max(Λ))                                 O(m)
ffunc
```

Algoritmo 5: Algoritmo para la función de costo Ω

La complejidad del algoritmo para el calculo de la función de costo Ω viene dada por la función **obtener_lambda** y es de $O(|V|^5)$ y permite encontrar $\pi(\Gamma, e) \forall e \in E \wedge E \in R \wedge R \in \Gamma$ de la siguiente forma:

1. Obtiene $\forall i, j : (i, j) \in E \wedge i \neq j$.
2. Dado un enrutamiento $R_r : 1 \leq r \leq |V|(|V|-1)$ itera por cada enrutamiento con el enlace compuesto por (i, j).
3. $\forall c_{i,j}^{r,k}$ se obtiene la congestión para cada enlaces e compuesto por los vértices (i, j) en el k-ésimo camino del r-ésimo enrutamiento, obteniendo la cantidad de caminos que pasan por (i, j).
4. Si la congestión$\neq 0$ se incluye en el vector Λ.
5. Se repite desde el paso 1 hasta haber completado todos los enlaces $(i, j) : i \neq j$.

```
func obtener_lambda(val r: Enrutamiento, val n: entero) : arreglo [1···m] de entero                  
    var:                                                                                             
        Λ : arreglo [1···m] de entero                                                                
        congestion, i,j : entero                                                                     
        u,v : Vertice                                                                                
    inicio:                                                                                          
        j ← 1                                                                          O(1)          
        u ← 1                                                                          O(1)          
        congestion ← 0                                                                 O(1)          
        para v ← 1 hasta n*n en 1 hacer                                                O(|V|^2)      
            si (u ≠ v%n) entonces                                                      O(1)          
                para i ← 1 hasta n*(n-1) en 1 hacer                                    O(|V|(|V|-1)) 
                    congestion ← verficar_enlace(r[i],u,v)                             O(|V|)        
                fpara                                                                              
                si (congestion≠0) entonces                                             O(1)          
                    Λ[j] ← congestion                                                  O(1)          
                    j ← j+1                                                            O(1)          
                fsi                                                                                
                congestion ← 0                                                         O(1)          
                si ((v+1)%n == 0) entonces                                             O(1)          
                    u ← u+1                                                            O(1)          
                fsi                                                                                
            fsi                                                                                    
        fpara                                                                                      
        retonar(Λ)                                                                     O(1)          
ffunc
```

Algoritmo 6: Algoritmo para obtener Λ

La complejidad del algoritmo que permite calcular Λ se puede estudiar a partir de la función **verificar_enlace** la cual posee una complejidad $O(|V|)$, esta función sera llamada $|V|(|V|-1)$, por lo que en el ciclo mas interno la complejidad viene siendo $O(|V|^2(|V|-1))$, esto puede ser reducido a $O(|V|^3)$; por ultimo, en el ciclo mas externo la complejidad viene dada por $O(|V|^2)$, esta complejidad sera multiplicada por la complejidad del ciclo interno obteniendo que la complejidad del algoritmo presentado es de $O(|V|^5)$

4.3.3. Operador de Selección de Padres Ψ

Dada una población $P \in \Re$ y una función de costo Ω, se define $F \subset P \wedge |F| = 2$ como el conjunto de par de individuos denominados padres tal que:

$$F = \{min_1\{\Omega(p_i)\}, min_2\{\Omega(p_i)\}\} : p_i \in P \wedge 1 \leq i \leq \mu$$

El **Algoritmo. 7** sera el encargado de la selección del conjunto F dentro de todo el conjunto P, para esto el algoritmo lleva a cabo los siguientes pasos:

1. Se inicia con el conjunto $F = \{p_1, p_2\} : p_1, p_2 \in P$
2. Se almacena el costo de $\{p_1, p_2\}$ a través de la función Ω.
3. Se toma el individuo $p_i : 3 \leq i \leq \mu$ y se calcula su función de costo Ω.
4. En caso de que $\Omega(p_i) < \Omega(p_1) \vee \Omega(p_i) < \Omega(p_2)$ se realiza $p_1 = p_i \vee p_2 = p_i$.
5. Se repite el proceso desde el paso 3 hasta que se haya explorado todo el conjunto P.

El operador Ψ consta de un ciclo en el cual se observa que ejecuta μ a sus operaciones internas, dentro de estas operaciones la que posee una complejidad elevada es la relaciona con la función de costo Ω siendo así su complejidad $O(|V|^5$, al estar esta operación dentro de un ciclo podemos decir que se ejecuta las veces que se repita el ciclo, dado el análisis indica que la complejidad del algoritmo viene dada por $O(\mu|V|^5)$.

```
func Ψ(val p: Poblacion, val n: entero) : Poblacion                                  
    var:                                                                             
        p, padres: Poblacion                                                         
        r1, r2: Enrutamiento                                                         
        costo_r1, costo_r2, costo_aux, i: entero                                     
    inicio                                                                           
        r1 ← p[1]                                                          O(1)      
        r2 ← p[2]                                                          O12)      
        costo_r1 ← Ω(r1,n)                                                 O(|V|^5)  
        costo_r2 ← Ω(r2,n)                                                 O(|V|^5)  
        costo_aux ← 0                                                      O(1)      
        para i ← 3 hasta μ en 1 hacer                                      O(μ)      
            costo_aux ← Ω(p[i])                                            O(|V|^5)  
            si (costo_aux < costo_r1) entonces                             O(1)      
                r1 ← p[i]                                                  O(1)      
                costo_r1 ← costo_aux                                       O(1)      
            sino                                                                     
                si (costo_aux <= costo_r2) entonces                        O(1)      
                    r2 ← p[i]                                              O(1)      
                    costo_r2 ← costo_aux                                   O(1)      
                fsi                                                                  
            fsi                                                                      
        fpara                                                                        
    insertar_poblacion(padres, r1)                                         O(1)      
    insertar_poblacion(padres, r2)                                         O(1)      
    retornar(padres)                                                       O(1)      
fin
```

Algoritmo 7: Algoritmo para obtener los padres de una población Ψ

4.3.4. Operador de Generación de Hijos Φ

Sea $F \subset P$ tal que $F = \{p_1, p_1\}$ representa los padres seleccionados por el operador Ψ, se define la operación $p_1 \times p_2 \rightarrow p_{fl}$ tal que p_l representa el l-esimo hijo de p_1 y p_2 en el cual cada $c_{i,j}^{r,k} : 1 \leq r \leq 2$ representa un camino de unos de los padres en el cual $P(c_{i,j}^{1,k} \vee c_{i,j}^{2,k}) = 0{,}5$.

El **Algoritmo. 8** hace referencia al operador que permite generar una nueva población dado un par de individuos con una complejidad $O(\mu|V|^2)$ y puede definirse como:

1. Se inicia una nueva población.
2. Se realiza el cruce entre los padres $p_1 \times p_2 \rightarrow p_r$ y se le conoce como hijo.
3. Se agrega el hijo generado a la nueva población.
4. Se repite el proceso a partir del paso dos hasta que se haya realizado $\mu - 2$ veces.
5. Se devuelve la nueva población.

```
func Φ(val padres: Poblacion, val g: Grafo, var μ: entero) : Poblacion
    var:
        nueva_poblacion : Poblacion
        r : Enrutamiento
        i : entero
    inicio
        para i ← 1 hasta μ en 1 hacer                                  O(μ)
            nueva_poblacion ← cruce(padres, g)                         O(|V|(|V|−1))
            insertar_poblacion(nueva_poblacion, r)                     O(1)
        fpara
        retornar(nueva_poblacion)                                      O(1)
ffunc
```

Algoritmo 8: Algoritmo para generar una población de hijos Φ

Para obtener la complejidad del operador Φ se debe hacer referencia al ciclo principal el cual ejecuta μ veces la funcion de cruce la cual posee una complejidad $O(|V|(|V|-1))$, por lo que se puede decir que la complejidad del operador viene dada por $O(\mu|V|(|V|-1))$.

$$O(\mu|V|(|V|-1))$$

$$O(\mu(|V|^2-|V|))$$

$$O(\mu|V|^2)$$

El **Algoritmo. 9** permite generar un hijo dado unos padres $(p_1, p_2) : p_1, p_2 \in F$ con una complejidad $O(|V|(|V|-1))$, para realizar la operación se siguen los siguientes pasos:

1. Se genera una probabilidad aleatoria entre 0 y 1.
2. Si la probabilidad generada es menor a 0,5 se toma el camino $c_{i,j}^{1,k} : c_{i,j}^{1,k} \in p_1$, en caso contrario $c_{i,j}^{2,k} : c_{i,j}^{2,k} \in p_2$.
3. El paso 2 repite $|V|(|V|-1)$ veces hasta generar un nuevo individuo.
4. Se retorna el nuevo individuo generado.

```
func cruce(val padres: Poblacion, val g: Grafo) : Enrutamiento                    
    var:                                                                          
        r : Enrutamiento                                                          
        c : Camino                                                                
        d,i : entero                                                              
    inicio                                                                        
        d ← numero_vertices(g) // n = |V|                                  O(1)   
        para i ← 1 hasta n*(n-1) en 1 hacer                          O(|V|(|V|-1))
            si (generar_probabilidad < 0,5) entonces                       O(1)   
                insertar_enrutamiento(padres[1][i])                        O(1)   
            sino                                                           O(1)   
                    insertar_enrutamiento(padres[2][i])                    O(1)   
            fsi                                                                   
        fpara                                                                     
        retonar(r)                                                         O(1)   
ffunc
```

Algoritmo 9: Algoritmo para generar un hijo (función de cruce)

4.3.5. Operador de Mutación Υ

El operador Υ le asigna una probabilidad aleatoria a todo p_i dentro de la población de hijos, dado una probabilidad α si $P(c_{i,j}^{r,k}) < \alpha$ se sustituye el camino $c_{i,j}^{r,k}$ por uno nuevo, los pasos del algoritmo vienen dados por:

1. Se selecciona un individuo de la población de hijos.
2. Se selecciona un camino del individuo seleccionado.
3. Se genera una probabilidad aleatoria entre 0 y 1.
4. En caso de que la probabilidad generada sea menor que *alpha* se genera un nuevo camino.
5. Se repite desde el paso 1 hasta que se haya recorrido todos los caminos de todos los individuos.

```
func ϒ(val nueva_poblacion: Poblacion, val g: Grafo, val α : real, val μ : entero) : Enrutamiento
    var:
        i,j,d : entero
    inicio
        d ← obtener_vertices(g)                                              O(1)
        para i ← 1 hasta μ en 1 hacer                                        O(μ)
            para j ← 1 hasta n*(n−1) en 1 hacer                              O(|V|(|V|−1))
                si (generar_probabilidad < α) entonces                       O(1)
                    nueva_poblacion[j] ← generar_camino(g,nueva_poblacion[i][j])   O(|V|+|E|)
                fsi
            fpara
        fpara
        retonar(nueva_poblacion)                                             O(1)
ffunc
```

Algoritmo 10: Algoritmo de mutacion ϒ

Tomando en cuenta que la complejidad para generar un camino viene dada por $O(|V|+|E|)$ y se deben generar $\mu|V|(|V|-1)$ veces, se tiene que la complejidad viene dada por:

$$O(|V|+|E|(\mu|V|(|V|-1)))$$

$$O(|V|^2(\mu(|V|^2-|V|)))$$

$$O(\mu(|V|^4-|V|^3)))$$

$$O(\mu|V|^4)$$

Concluyendo con el análisis del **Algoritmo. 10** se obtuvo que la complejidad de aplicar el operador de mutación ϒ a toda una población de μ individuos viene dada por $O(\mu|V|^4)$.

4.3.6. Operador de Nueva Población Θ

Sea $F \subset P_i : 1 \leq i \leq \beta \wedge |F| = 2$ el conjunto de los padres seleccionados en la población i-ésima y sea $P_F : |P_F| = \mu - 2$ el conjunto de la población obtenida mediante el cruce entre los padres, se define la población $P_{i+1} = F \cup P_F : |F| + |P_F| = \mu$. El algoritmo consiste en:

1. Generar una nueva población $P_{i+1} = \{\}$.
2. Se realiza la siguiente operación $P_{i+1} = P_F \cup P_i$.
3. Se combinan los padres junto al conjunto obtenido en el paso 2 tal que $P_{i+1} = P_{i+1} \cup F$
4. Se retorna la nueva población $P_{i+1} : |P_{i+1}| = |F| + |P_F|$.

```
func Θ(val poblacion_hijos: Poblacion, var padres: Poblacion) : Poblacion
    var:
        p: Poblacion
    inicio
        p ← poblacion_hijos                                  O(1)
        insertar_poblacion(p, padres[1])                     O(1)
        insertar_poblacion(p, padres[2])                     O(1)
        retornar(p)                                          O(1)
ffunc
```

Algoritmo 11: Algoritmo del operador nueva población Θ

Dado que el algoritmo de nueva población consiste en la unión de dos conjuntos, todas las operaciones son constante por lo que el operador Θ posee una complejidad constante $O(1)$.

4.3.7. Función de Mejor Enrutamiento η

Sea P el conjunto de soluciones luego de β generaciones, se dice que el mejor enrutamiento $p_i \in P$ es aquel donde $min(\Omega(p_i))$, el **Algoritmo. 12** consiste en:

1. Se selecciona el primer individuo p_1 de la población como el mejor de acuerdo a la función de costo Ω y se almacena como el mejor candidato r.
2. Se verifica si el individuo $p_i : 2 \leq i \leq \beta$ actual es mejor que r.

3. Se repite el paso 2 μ veces.

4. Se devuelve $r : min(\Omega(r))$.

```
func η(val p: Poblacion, val μ: entero) : Enrutamiento                    
    var:                                                                  
        r: Enrutamiento                                                   
        mejor_r, actual_r: real                                           
        i: entero                                                         
    inicio                                                                
        mejor_r ← Ω(p[1])                                       O(|V|^5)  
        r ← p[1]                                                O(1)      
        para i ← 2 hasta μ en 1 hacer                           O(μ)      
            actual_r ← Ω(p[i])                                  O(|V|^5)  
            si (actual_r < mejor_r entonces                     O(1)      
                mejor_r ← actual_r                              O(1)      
                r ← p[i]                                        O(1)      
            fsi                                                           
        fpara                                                   O(1)      
        retornar(r)                                                       
ffunc                                                                     
```

Algoritmo 12: Algoritmo de Mejor Enrutamiento

La complejidad del algoritmo para obtener el mejor enrutamiento de la población η viene dada por el ciclo y la cantidad de veces que se llama a la funcion de costo Ω, la complejidad de la funcion de costo viene dada por $O(|V|^5)$, dado que esta funcion es llamada μ la complejidad del algoritmo sera de $O(\mu|V|^5)$.

4.3.8. Algoritmo Genético

La complejidad del **Algoritmo. 13** propuesto posee complejidad $O(\beta\mu|V|^5)$ y de forma explicita consiste en los siguientes pasos:

1. Se genera la población inicial utilizando el operador Ξ.

2. Se selecciona el mejor par de individuos de la población a través del operador Ψ.

3. Se genera una nueva población usando el operador Φ.

4. Se le aplica el operador de mutación Υ a la nueva población generada.

5. Se construye la población combinando la nueva población y los padres seleccionados a través del operador Θ.

6. Se vuelve al paso 2 hasta se que hayan cumplido β generaciones.

7. Se selecciona el mejor individuo de la población a través de la función η.

```
func algoritmo_genetico(val G: Grafo, val β:entero, val α: real, val μ entero) : Enrutamiento
    var:
        poblacion, nueva_poblacion, padres : Poblacion
        generacion: entero
    inicio
        poblacion ← Ξ(G, μ)                                          O(μ(|V|^4))
        para generacion ← 1 hasta β en 1 hacer                       O(β)
            padres ← Ψ(poblacion)                                    O(μ|V|^5)
            nueva_poblacion ← Φ(padres, g, μ − 2)                    O(μ|V|^2)
            nueva_poblacion ← ϒ(nueva_poblacion, g, α)               O(μ|V|^4)
            poblacion ← Θ(nueva_poblacion, padres)                   O(1)
        fpara
        retornar(η(población))                                       O(μ|V|^5)
ffunc
```

Algoritmo 13: Algoritmo Genético

4.4. Parámetros de Entonación

- β: Representa el numero de generaciones que algoritmo realizara antes de finalizar su ejecución, $\beta \in \mathbb{N}$.
- μ: Representa la cantidad de individuos en la población P, $\mu \in \mathbb{N}$.
- α : Representa la probabilidad que posee un individuo p de mutar, cabe considerar que la probabilidad de mutación debe ser pequeña $0 \leq \alpha \leq 0{,}2$.

5. Experimentos

5.1. Configuración de los Experimentos

El **Algoritmo. 13** se ha propuesto para calcular el índice de transmisión de enlaces sobre un grafo y posee una complejidad de $O(\beta\mu|V|^5)$. El primer paso a realizar es la entonación de los parámetros $\alpha = 0{,}02, \beta = 30, \mu = 30$ los cuales representan la probabilidad que tiene un individuo de mutar, la cantidad de generaciones y la cantidad de individuos de la población respectivamente.

El primer paso a realizar para llevar a cabo el experimento que permita probar y analizar el rendimiento de la solución algorítmica planteada en la presente investigación es llevar la implementación del algoritmo, para esto se utilizara el lenguaje de programación C++. Las razones para la elección del lenguaje viene dada por diversas razones:

- Facilidad de crear y utilizar diferentes tipos de datos.
- Permite manejar el uso de memoria.
- Posee menos errores debido a que la sintaxis y chequeo de errores es mucho mas estricta.
- Rapidez.

La implementación del algoritmo propuesto fue puesta a prueba haciendo uso de grafos generados automáticamente en las topologías de brujin, hipercubo, estrella, rueda y ciclo, para así poder probar que tan solido y eficiente es la solución propuesta.

5.1.1. Plataforma Computacional

Para realizar las pruebas de la implementación se utilizo una instancia de los servidores de *Amazon Web Services (AWS)* como plataforma computacional en la

cual se requirió instalar todo lo necesario para ejecutar programas hecho bajo C++, las característica son:

- **Memoria:** 36GiB
- **Procesador (GPU):**
  - **Modelo:** *Intel(R) Xeon(R) Platinum* 8175M CPU @ 2.50GHz
  - **Vendedor**: *Intel Coporation*
  - **Número de núcleos**: 8
  - **Memoria Caché: 38**.5MB L3 Cache
  - **Reloj:** 100MHz
  - **Capacidad:** 3500MHz
  - **Número de Hilos (*Threads*):** 4
  - **Repertorio de Instrucciones:** x86-64
- **Descripción del Computador:**
  - **Producto**: m5.2xlarge
  - **Vendedor**: *Amazon* EC2
  - **Serial**: ec2c5afc-c0aa-29cc-dea1-76f8f4daf810
  - **Anchura**: 64 bits
  - **Capacidades**: smbios-2.7 dmi-2.7 smp vsyscall32
  - **Configuración**: uuid=FC5A2CEC-AAC0-CC29-DEA1-76F8F4DAF810
- **Sistema Operativo**: Ubuntu 18.04
- **Lenguaje de Programación:** C++11 junto a g++7.4.0

Adicionalmente se trabajo con un controlador de versiones, tal como es el caso de *Git* usando la plataforma web *GitLab*, el repositorio creado puede encontrarse en `https://gitlab.com/Kinozuko/teg.git` y posee la siguiente estructura:

- ***test_cases***: Se encuentran todos los casos de prueba utilizados con el algoritmo para los grafos ciclo, rueda, de brujin, estrella e hipercubo.
- ***makefile***: Archivo para compilar el código y obtener el ejecutable.
- ***main.cpp***: Código de la solución algorítmica presentada.
- ***results***: Almacena los resultados de todas las pruebas.
- ***README***: Información sobre el repositorio e instrucciones de ejecución.

5.1.2. Casos de Prueba

El formato de los archivos utilizados en los grafos generados posee una variación del formato trivial de grafos (*TGF*) en el cual se especificar un parte de vértices$(v_i, v_j$ con la siguiente estructura:

$$v1, v2$$

$$v2, v3$$

$$v3, v4$$

$$\vdots, \vdots$$

$$v_d, v_d$$

La cantidad de casos de prueba varia de acuerdo al tipo de grafo, para el experimento a realizarse de definieron los siguientes casos:

- **Ciclo**: Casos de 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70 y 80 vértices.
- **Estrella**: Casos de 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90 y 100 vértices.
- **Rueda**: Casos de 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90 y 100 vértices.
- **Hipercubo**: Casos de 4, 8, 16, 32, 64 y 128 vértices.

- **Brujin**: Casos de 3, 4, 5, 6, 7, 8, 9, 10, 20, 27, 31, 49, 64, 81 y 100 vértices.

Para obtener un mejor panorama de los resultados cada prueba fue realizada 3 veces con el fin de estudiar la convergencia del índice de transmisión sobre enlace y la consistencia con respecto al tiempo de ejecución de forma que permita reproducir los resultados obtenidos en este trabajo.

5.1.3. Métricas de Evaluación

Se definieron dos métricas, la primera hace referencia al índice de transmisión de enlaces sobre un grafo otorgado por el algoritmo mientras que la otra consiste en el tiempo de ejecución de cada prueba la cual es medida en microsegundos. Estas métricas serán las encargadas de observar la consistencia de los casos de prueba para las distintas topologías.

5.2. Análisis de Resultados

5.2.1. Hipótesis Inicial

Una vez definido las configuraciones necesarias para realizar el experimento se propone una hipótesis inicial la cual sera evaluada a través de los resultados experimentales para así poder concluir sobre la hipótesis, esta hipótesis puede definirse como H_0: La solución obtenida a través del algoritmo genético resuelve el índice de transmisión sobre enlaces de forma consistente y en tiempo razonable.

La validación de la hipótesis H_0 vendrá dada por las métricas definidas anteriormente, estas son el índice de transmisión y el tiempo de ejecución. Al ejecutar casos de prueba con distintos parámetros de entonación se compararan sus índices, aquel con el índice menor sera una mejor solución a nivel de resultado; si la diferencia entre los índices es pequeña pero el tiempo de ejecución es muy grande, la solución con el menor tiempo de ejecución sera considerada mejor a nivel experimental, por ultimo la mejor solución serán aquellos con el mejor índice obtenido.

Una vez concluido los casos de prueba los cuales fueron ejecutados tres veces cada uno, se agruparon los resultados por tipo de grafo, en donde los parámetros del algoritmo consistieron en una probabilidad de mutación $\alpha = 0{,}02$, una población de $\mu = 30$ individuos los cuales evolucionaran $\beta = 30$ generaciones.

5.2.2. Brujin

El primer experimento fue realizado a los grafos pertenecientes a la topología Brujin, durante la evaluación solo se realizaron los experimentos para los vértices menores a 100 siendo este ultimo el experimento final. El comportamiento de los resultados obtenidos (ver **Fig. 2**) se puede observa que se mantiene en un orden polinomial con una variación en el tiempo de ejecución.

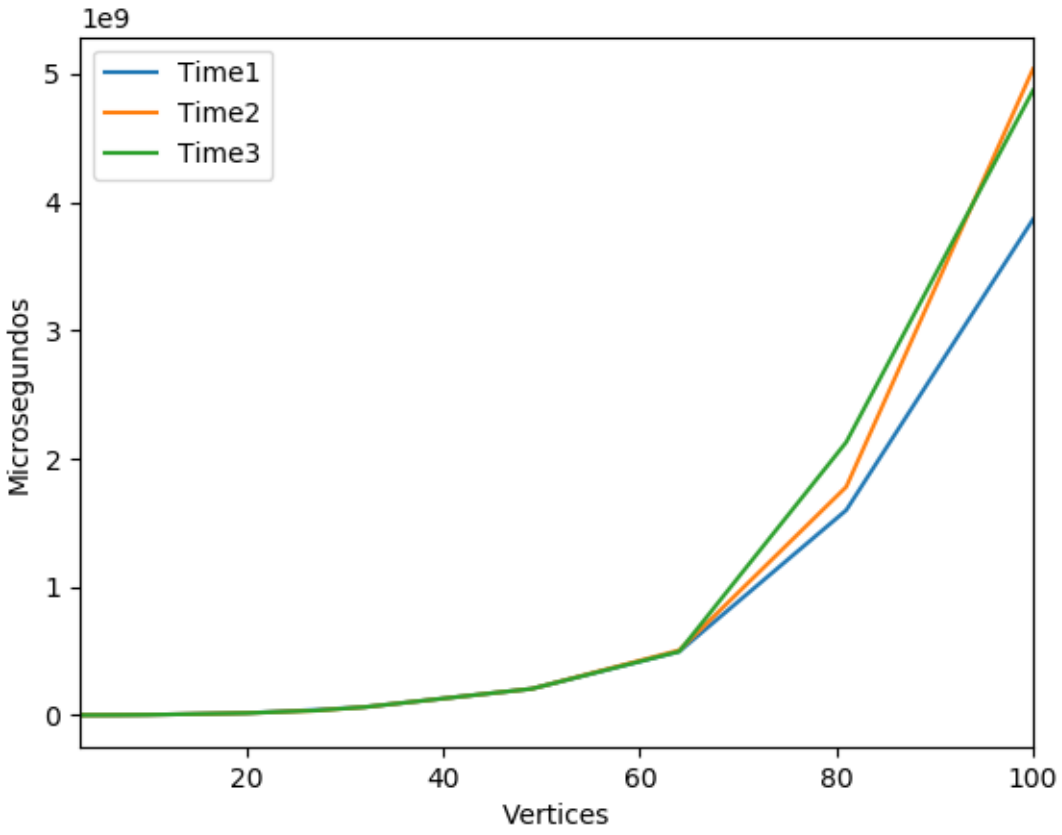

Figura 2: Rendimiento para los Grafos de Brujin

Detallando los resultados obtenidos (ver **Tabla. 1**), se logra observar que el tiempo de ejecución se mantiene muy parecido para la mayoría de los casos, exceptuando los casos con vértices igual a 27, 81 y 100, en el cual se puede observar la mayor variación. A nivel de tiempo respuesta se puede concluir que el algoritmo es constante para la mayoría de los casos. Con respecto a los índices obtenidos en todos los casos se observa que la variación es mínima, siendo esta de ± 1 para los casos de vértices igual a 27, 49, 64 y 81, mientras que la variación es de ± 3 cuando hay 32 vértices, con esto podemos decir que a pesar de que el calculo del índice no es constate, la variación es mínima para la mayoría de los casos.

Vértices	**Índice 1**	**Tiempo 1**	**Índice 2**	**Tiempo 2**	**Índice 3**	**Tiempo 3**
3	2	195925	2	195421	2	195196
4	3	388146	3	387358	3	387893
5	2	649613	2	649151	2	650731
6	2	986347	2	985775	2	984401
7	2	1398783	2	1398950	2	1398487
8	11	1880927	11	1879721	11	1878110
9	6	2451094	6	2452274	6	2451117
10	2	3112530	2	3117728	2	3110411
20	2	16220467	2	16205841	2	16212146
27	27	40992826	27	36405408	26	36415032
32	66	63106673	69	63055958	66	63120336
49	20	205857463	20	205850801	19	205669028
64	23	496439266	22	505791153	23	496613508
81	25	1598614755	26	1780450093	26	2128858483
100	28	3865211237	28	5037326886	28	4870138238

Tabla. 1: Resultados de Grafos Brujin

5.2.3. Rueda

Al igual que el experimento realizado a los grafos Brujin, solo se realizaron pruebas para los vértices menores a 100. En la gráfica (ver **Fig. 3**) se observa que el comportamiento de los resultados entre la cantidad de vértices y tiempo de ejecución se mantiene constante exceptuando una variación notable en la segunda prueba (curva naranja).

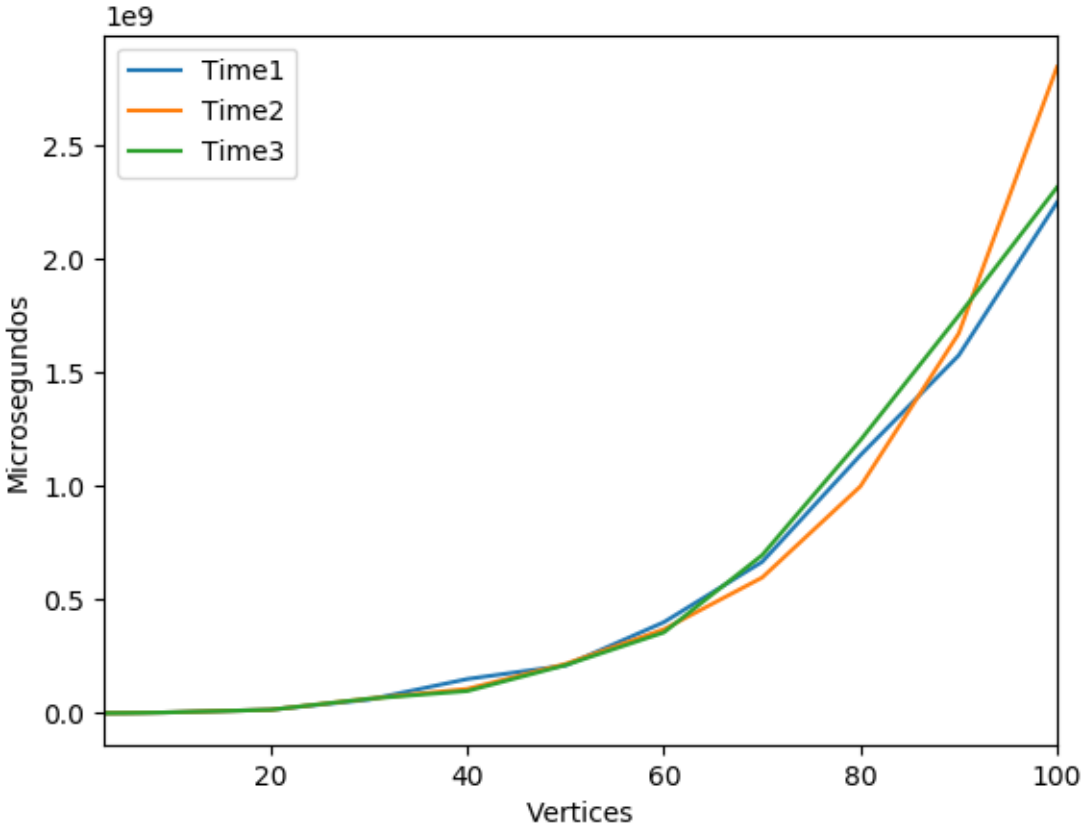

Figura 3: Rendimiento para los Grafos de Rueda

En la **Tabla. 2** se observa los tiempos de ejecución así de como el valor del índice encontrado. Con respecto a la consistencia en tiempo de ejecución se puede encontrar variación para el caso donde hay 30, 40, 60, 80, 90 y 100, con esto podemos decir a pesar de que la consistencia se mantuvo para las dos primeras pruebas no fue así el caso para la ultima. Con respecto a la consistencia de los índice se observa una variación mínima de ± 1 para los casos con 8, 40 y 90 vértices por lo que el índice obtenido para este tipo de grafos es consistente.

5.2.4. Estrella

El la **Fig. 4** se observa que las curvas azul y naranja son muy similares, mientras que para la curva verde existe una variación. Por otro lado las gráficas para las tres pruebas fueron muy similares cuando se posee menos de 80 vértices.

Vértices	Índice 1	Tiempo 1	Índice 2	Tiempo 2	Índice 3	Tiempo 3
3	2	265447	2	265345	2	265684
4	2	526583	2	528375	2	528084
5	4	848434	4	848273	4	848621
6	4	1289388	4	1289061	4	1288623
7	6	1833570	6	1835855	6	1834682
8	7	2538481	6	2465804	7	2466856
9	9	3213671	9	3209120	9	3210029
10	11	4088469	11	4090134	11	4092749
20	32	15282395	32	15270670	33	15275742
30	53	59975334	53	63985843	53	63913846
40	73	150619373	73	107287038	72	97720138
50	93	210279321	93	217152143	93	211988998
60	113	400818824	113	369664149	113	355744677
70	133	666474322	133	597395075	133	695951502
80	153	1136276445	154	999129952	153	1202292602
90	173	1575577906	174	1672204824	174	1751143351
100	194	2248983131	194	2845242871	194	2315538184

Tabla. 2: Resultados para los Grafos Rueda

Los tiempos de ejecución así como el índice obtenido para los grafos estrella pueden visualizarse en la **Tabla. 3**. La consistencia con respecto al tiempo de ejecución fue bastante notoria para la mayoría de los vértices en las tres pruebas, cuando los vértices son 90 y 100 la variación con respecto a los tiempos es mucho mas notable. Los índice obtenidos fueron consistentes para todas las pruebas, se puede decir que el algoritmo propuesto es consistente con respecto al índice para esta topología de grafos con máximo 100 vértices.

5.2.5. Ciclos

La gráfica (ver **Fig. 5**) representa la relación entre la cantidad de vértices y el tiempo de ejecución de cada una de las pruebas realizadas. Las curvas azul y naranja representa las dos primeras pruebas y se observa que su comportamiento es muy similar una a la otra con una pequeña variación a medida que los vértices se van incrementando. Por otro lado, la curva verde representa la tercera prueba y es muy similar a las otras dos con una variación brusca a partir de los 60 vértices.

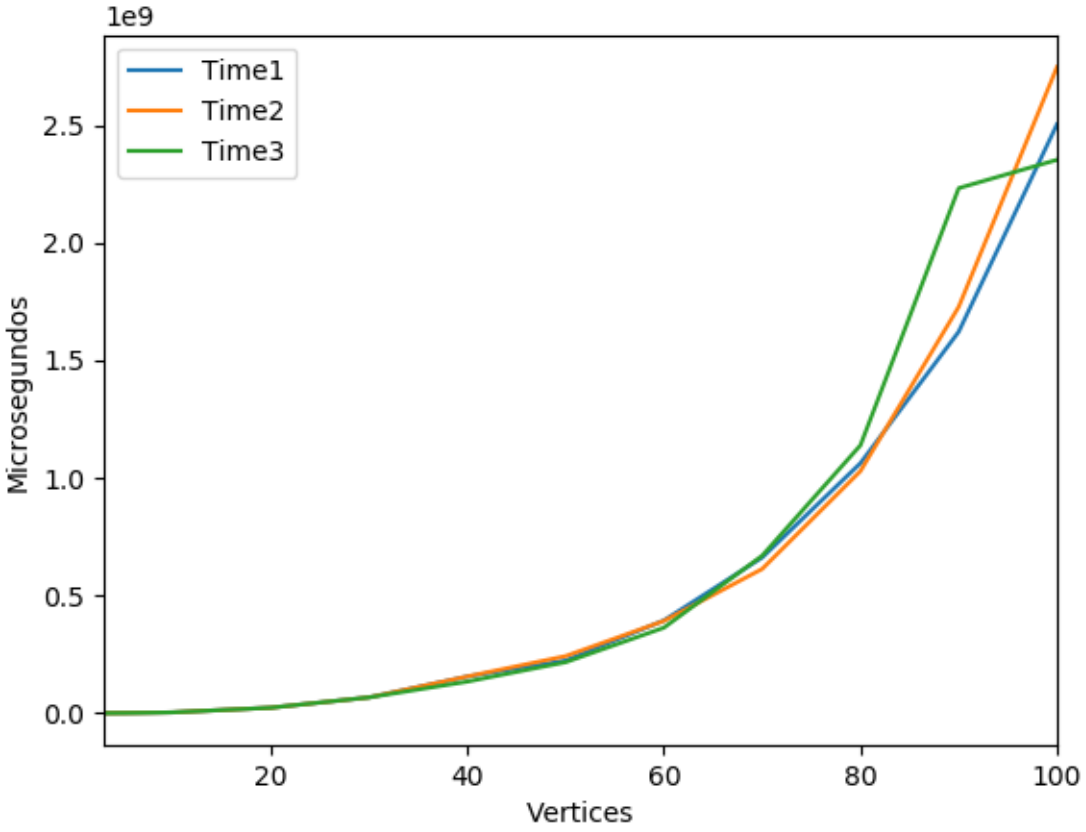

Figura 4: Rendimiento para los Grafos Estrella

Los valores obtenidos en las tres pruebas pueden detallarse en la **Tabla. 4**, se logra observar que los tiempos de ejecución para os grafos ciclo son muy similares para las tres pruebas realizadas, la diferencia se encuentra cuando se poseen 70 vértice debido a que en la tercera prueba con esta cantidad el tiempo fue duplicado con respecto a las dos pruebas anteriores, por lo que se pude decir que la consistencia en tiempo de ejecución es constante para todos los casos menores a 60 vértices. Con respecto a los valores obtenidos de los índices se observa una variación de ± 1 para 20, 40, 50 y 60 vértices, por lo que la consistencia en los resultados también es constante con un error de ± 1.

Vértices	Índice 1	Tiempo 1	Índice 2	Tiempo 2	Índice 3	Tiempo 3
3	6	387816	6	387186	6	387187
4	8	646398	8	646156	8	645007
5	10	973971	10	973958	10	973834
6	12	1382096	12	1383475	12	1382395
7	14	1861944	14	1861864	14	1859221
8	16	2414127	16	2416875	16	2412381
9	18	3041859	18	3045663	18	3044255
10	20	3767451	20	3767496	20	3765912
20	40	23686590	40	23487740	40	23482465
30	60	67368740	60	67353252	60	67347424
40	80	156070698	80	156366449	80	134503932
50	100	224304127	100	242821579	100	217075244
60	120	394520019	120	392597190	120	363702496
70	140	662061834	140	613523798	140	669885272
80	160	1063224937	160	1029483054	160	1139359521
90	180	1622177040	180	1728070916	180	2232595918
100	200	2503834000	200	2747439921	200	2353072326

Tabla. 3: Resultados de Grafos Estrella

5.2.6. Hipercubo

En la gráfica presentada en la **Fig. 6**, se puede observar que los resultados de las pruebas dos y tres (curvas naranja y verde respectivamente) se encuentra prácticamente solapadas, lo cual indica que los resultados entre estas son muy parecidos. Adicionalmente se puede observar que el tiempo de ejecución posee un incremento considerable a medida que se van agregando vértices.

Los resultados obtenidos para el grafo hipercubo pueden observarse en la **Tabla. 5**, en el análisis de consistencia en tiempo de ejecución se observa valores muy similares, menos para la prueba uno cuando se posee 32 y 64 vértices, sin embargo, los tiempos de ejecución para un grafo de 128 vértices se mantiene consistente. Por otro lado, la consistencia de los resultados se observa que las variaciones comienzan con 32 y 64 vértices con una variación mínima de ± 1 mientras que para 128 vértices la variación es de ± 2.

Vértices	Índice 1	Tiempo 1	Índice 2	Tiempo 2	Índice 3	Tiempo 3
3	2	194644	2	194524	2	194469
4	4	386804	4	387175	4	386863
5	6	645896	6	645939	6	645493
6	9	979935	9	980736	9	979655
7	12	1391575	12	1391789	12	1392739
8	16	1891229	16	1890152	16	1887480
9	20	2455604	20	2455748	20	2457025
10	25	3137245	25	3136684	25	3137095
20	101	18966320	100	19058819	101	18956247
30	226	75875966	226	75897718	226	75895418
40	402	240937637	401	240944624	401	242266774
50	626	631288833	627	626569530	626	626643360
60	902	1426612219	902	1515527236	903	1510785615
70	1227	2911928291	1227	2927355474	1227	4358172511
80	1602	5737932690	1602	5738268225	1602	5738094822

Tabla. 4: Resultados de Grafos Ciclo

Vértices	Índice 1	Tiempo 1	Índice 2	Tiempo 2	Índice 3	Tiempo 3
4	4	386134	4	386460	4	386667
8	9	1881917	9	1880491	9	1881789
16	18	9394253	18	9386232	18	9391668
32	37	77613988	37	88169206	38	88174514
64	75	725305852	74	647703959	75	648217479
128	146	9938501052	148	9928973500	147	9912544653

Tabla. 5: Resultados de Grafos Hipercubo

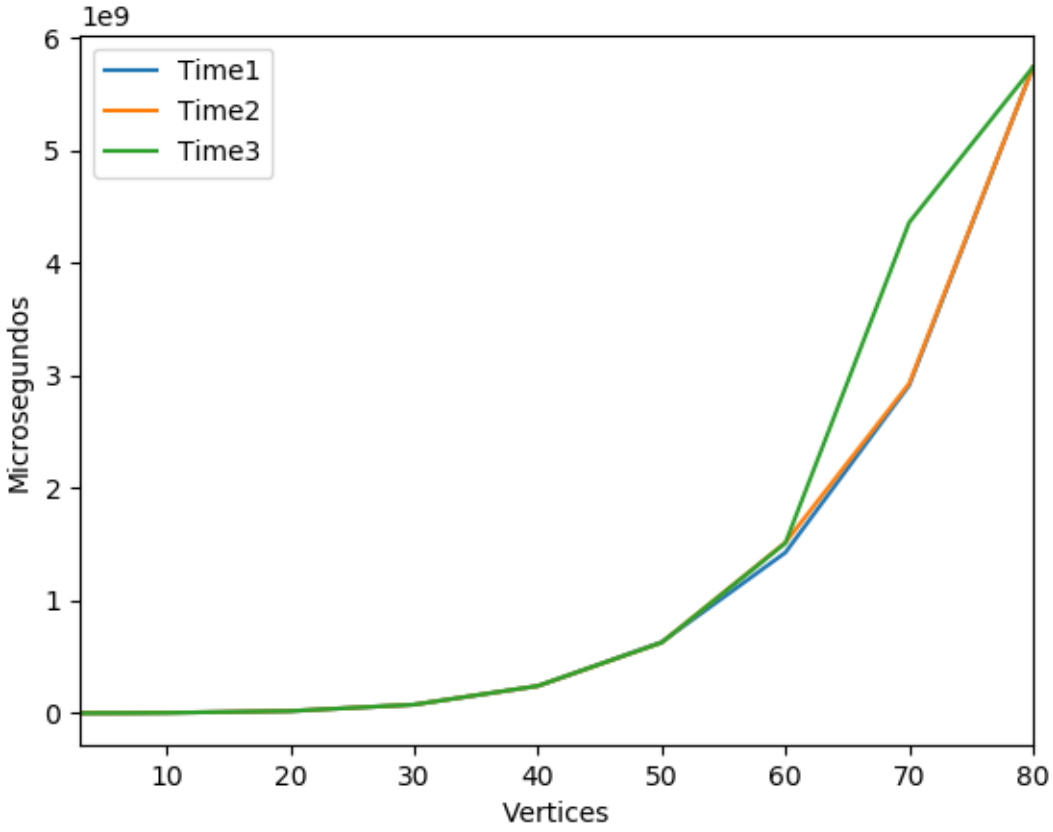

Figura 5: Rendimiento para los Grafos Ciclo

5.3. Entonación de Parámetros

Una vez concluido el análisis de los resultados, se selección un grafo de cada tipo donde la consistencia tanto en tiempo de ejecución y resultados sean buenas, esto con el fina de variar los parámetros de entonación α, β, μ, esto con el fin de observar las variaciones de los resultados ante distintas configuraciones de entonaciones. Las pruebas que se realizaron fueron hechas con $\alpha = 0{,}02$, $\mu = 30$ y $\beta = 30$, para el estudio se utilizo tres variantes distintas y se tomaron las pruebas con los tiempos de ejecución mas bajos, los parámetros de entonación fueron configurados de la siguiente forma:

- $\beta = 50$, $\mu = 15$ y $\alpha = 0{,}01$
- $\beta = 30$, $\mu = 20$ y $\alpha = 0{,}005$

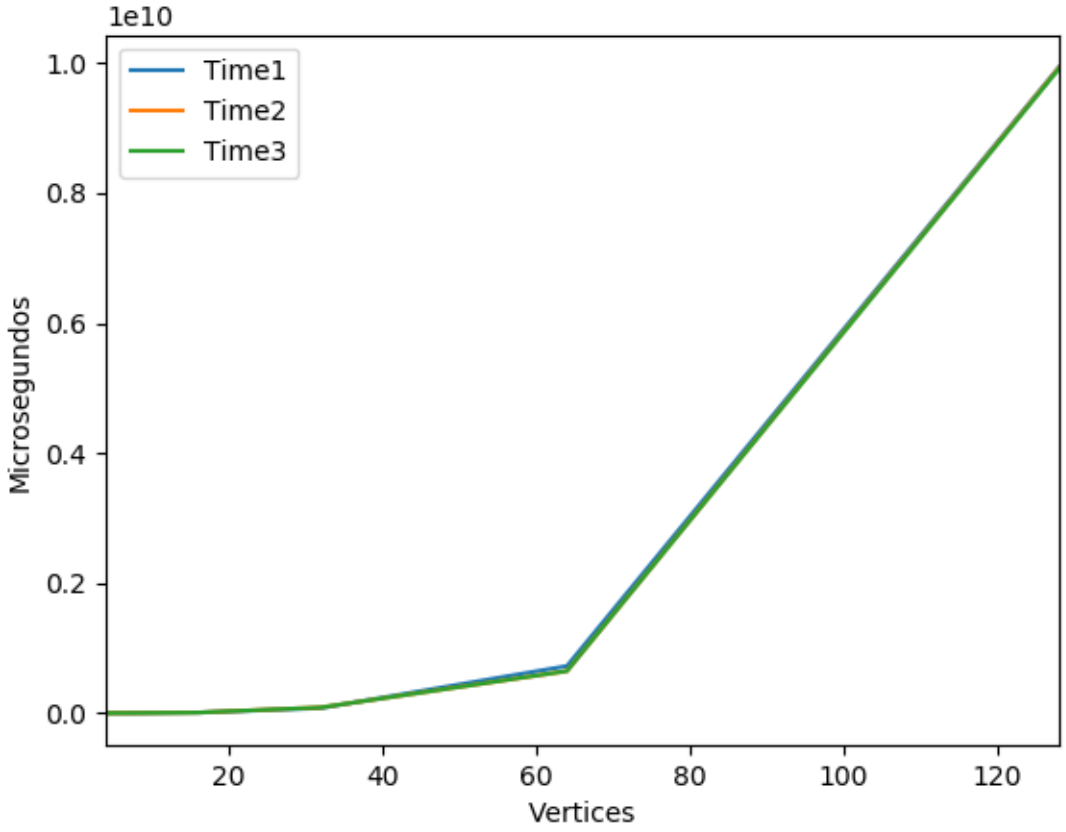

Figura 6: Rendimiento para los Grafos Hipercubo

- $\beta = 20$, $\mu = 60$ y $\alpha = 0{,}01$

Los grafos seleccionados para el análisis fueron los siguientes:

- Grafo ciclo de 50 vértices.
- Grafo de brujin de 64 vértices.
- Grafo hipercubo de 64 vértices.
- Grafo estrella de 60 vértices.
- Grafo rueda de 60 vértices.

Los resultados de las distintas configuraciones de los parámetros del algoritmo pueden verse en la **Tabla. 6**, se observo que para los grafos ciclo la confi-

β	μ	α	**Grafo**	**Vértices**	**Índice**	**Tiempo**	**Observaciones**
30	30	0.02	Ciclo	50	627	626569530	Original
50	15	0.01			627	498620984	Mejora en tiempo
30	20	0.005			626	406535065	Mejora en tiempo, desmejora en índice (-1)
20	60	0.015			626	838583084	Desmejora en tiempo e índice (-1)
30	30	0.02	Brujin	64	23	496439266	Original
50	15	0.01			23	378246821	Mejora en tiempo
30	20	0.005			23	316361561	Mejora en tiempo
20	60	0.015			22	699191170	Desmejora en tiempo e índice (-1)
30	30	0.02	Rueda	60	113	355744677	Original
50	15	0.01			113	272448845	Mejora en tiempo
30	20	0.005			113	225231503	Mejora en tiempo
20	60	0.015			113	482359150	Desmejora en tiempo
30	30	0.02	Estrella	60	120	363702496	Original
50	15	0.01			120	282919959	Mejora en tiempo
30	20	0.005			120	233176385	Mejora en tiempo
20	60	0.015			120	495133941	Desmejora en tiempo
30	30	0.02	Hipercubo	64	74	647703959	Original
50	15	0.01			74	510091022	Mejora en tiempo
30	20	0.005			73	419148872	Mejora en tiempo y desmejora en el índice (-1)
20	60	0.015			72	885127830	Desmejora en tiempo e índice (-2)

Tabla. 6: Entonación de Parámetros

guración $\beta = 50, \mu = 15, \alpha = 0{,}01$ logra reducir el tiempo de ejecución mientras mantiene la consistencia de los resultados, por otro lado, la configuración $\beta = 30, \mu = 20, \alpha = 0{,}005$ también logra reducir el tiempo de ejecución mucho mas a costa de una desmejora mínima con respecto al índice, por lo que esta configuración es la mas adecuada para los grafos ciclo si se permite un margen de error ± 1.

Para los grafos bruijn la mejor configuración resulto ser $\beta = 30, \mu = 20, \alpha = 0{,}005$, ya que es la que mas logra reducir el tiempo de ejecución mientras mantiene la consistencia de los resultados. Para los grafo rueda la configuración realizada para los grafo brujin también resulta ser la mejor ya que también logra reducir el tiempo de ejecución manteniendo el valor del índice consistente, para los grafos estrellas también se observo este comportamiento bajo la misma configuración. Por ultimo para los grafos hipercubo la misma configuración resulta en una mejora considerable con respecto al tiempo de ejecución pero se produce un error mínimo de ± 1 al momento de obtener el índice.

6. Conclusiones y Recomendaciones

6.1. Conclusiones de la Investigación

En la investigación realizada se presento un algoritmo de optimización basado en algoritmos genéticos para la resolución del índice de transmisión sobre enlaces de un grafo. Los resultados obtenidos durante la parte experimental de la investigación resulto en una convergencia con respecto al calculo del índice de transmisión con diferentes tiempos de ejecución para grafos no mayores a 100 vértices, exceptuando el de ciclo dado que los tiempos de ejecución para este iban creciendo de forma casi exponencial a medida que los vértices se iban incrementando, por lo que los grafos ciclo fueron los tuvieron el rendimiento mas bajo .

Los resultados obtenidos para los grafos rueda, brujin y estrella obtuvieron valores muy similares con respecto al tiempo de ejecución e índice para una cantidad no mayor a 100 vértices, dado que el problema anterior a partir de esa cantidad de vértices los tiempos empezaban a crecer de manera exponencial, en los tres casos se observo que el algoritmo resuelve eficientemente el problema cuando se poseen menos de 60 vértices dado que a partir de este punto los tiempo de ejecución se empiezan a disparar.

El desempeño para los grafos mas complejos como los hipercubos se pudieron realizar hasta 128 vértices, siendo su comportamiento el mas consistente debido a que en las tres pruebas realizadas para grafos menores a 128 vértices resultaron ser muy similares entre si, al igual que en los casos anteriores, el comportamiento del eficiente con respecto al tiempo de ejecución son eficientes hasta un aproximado de 60 vértices, a partir de ese punto los tiempos de ejecución se disparan y mucho mas con los grafos hipercubo.

Por ultimo, los parámetros de entonación se realizo con diversas configuraciones para observar el comportamiento del algoritmo con distintos grafos, con lo cual se pudo observar que la cantidad de generaciones posee un peso mayor que la cantidad de individuos, cabe mencionar que la complejidad del algoritmo además de incrementar con la cantidad de vértices existentes, también se ve afectada por la cantidad de individuos y las generaciones que se realizan, por lo cual estos parámetros son un punta de partida relevante al momento de realizar este tipo de algoritmos.

6.2. Recomendaciones

En los algoritmos genéticos y los basados en optimización en general, los parámetros de entonación son uno de los aspectos mas importantes, por lo que es necesario probar distintas configuraciones para probar el rendimiento en diferentes casos para observar cuales son los que mejor se adapten a cada tipo de grafo, esto debido a que estos dos parámetros en particular poseen gran influencia en el rendimiento del mismo ya que no es lo mismo tener una población pequeña e ir evolucionándola muchas veces, que tener una población grande que evolucione poco.

Se recomienda no realizar las pruebas con grafos muy extensos en computadores sin la cantidad de memoria suficiente ya que pudiera ocurrir un desbordamiento de memoria, por lo que si se requiere realizar pruebas mas exhaustivas se requiere de maquinas con una alta capacidad en memoria, Cabe destacar que el tiempo de ejecución para todos los grafos probados poseen un incremento a partir de los 60 vértices y volviéndose casi exponencial luego de los 100.

Dado que no existen soluciones experimentales que resuelvan el problema planteado no hay un punto de comparación de los resultados, por lo que los resultados obtenidos durante la investigación serán un buen punto de partida para comparar el rendimiento de otros tipos de optimización, incluyendo versiones mejoradas del

presentado en la investigación, permitiendo así al trabajo presentado la posibilidad de ser un antecedente para los trabajos de investigación futuros.

6.3. Trabajos Futuros

Uno de los principales problemas del algoritmo es la alta capacidad de memoria requerida, por lo que uno de los trabajos a futuro de esta investigación seria utilizara una optimización a nivel de memoria para así poseer un mejor control y poder reducir así el consumo. Para lograr esto se puede cambiar algunas estructuras y redefinir la estructura principal que almacena todos los caminos por otras, pero esto requeriría cambios drásticos en el algoritmo.

Finalmente, otra de los trabajos futuros para la investigación planteada, es la utilizara concurrencia o paralelismo para poder trabajar el conjunto de población de manera divida y aplicar un proceso de torneo entre los individuos para la selección de los padres, de forma tal que se logre trabajar con conjuntos pequeños de la población al mismo tiempo.

Referencias

[1] M.C. Heydemann, J.C. Meyer, and D. Sotteau. On forwarding indices of networks. *Discrete Applied Mathematics*, 23(2):103 – 123, 1989.

[2] R. Saad. Complexity of the forwarding index problem. *SIAM Journal on Discrete Mathematics*, 6(3):418–427, 1993.

[3] Kenneth Sörensen. Metaheuristics – the metaphor exposed. *International Transactions of Operations Research*, In Press, 01 2013.

[4] S.J. Ruseel and P. Norvig. *Inteligencia Artificial, Un Enfoque Moderno*, page 132. 2004.

[5] Wenhong Tian. On the transformability of p and np problems. 02 2017.

[6] Milena Bogdanović. Overview of some solved np-complete problems in graph theory. *Advances and Applications in Mathematical Sciences*, 9, 01 2011.

[7] Jun-Ming Xu and Min Xu. The forwarding indices of graphs – a survey. *Opuscula Mathematica*, 33, 04 2012.

[8] Fan Chung, E. Coffman, M. Reiman, and B. Simon. The forwarding index of communication networks. *IEEE Transactions on Information Theory*, 33(2):224–232, March 1987.

[9] Abdelmadjid Bouabdallah and Dominique Sotteau. On the edge forwarding index problem for small graphs. *Networks*, 23(4):249–255.

[10] K Trabelsi, Marc Sevaux, Philippe Coussy, André Rossi, and Kenneth Sörensen. *Metaheuristics*. 01 2010.

[11] Fidias Arias Odón. *EL PROYECTO DE INVESTIGACIÓN 6a EDICIÓN*. 07 2012.

[12] Masoud Yaghini and Rahim Akhavan. Dimma: A design and implementation methodology for metaheuristic algorithms-a perspective from software development. *Int. J. Appl. Metaheuristic Comput.*, 1(4):57–74, October 2010.

[13] Amadís Martínez-Morales and Daniel Rosquete. Naspi: Una notación algorítmica estándar para programación. *Télématique: Revista Electrónica de Estudios Telemáticos, ISSN 8156-4194, Vol. 8, N^o. 3, 2009, pags. 55-74*, Vol 8, 09 2009.