Diseño de redes de telecomunicación con múltiples escenarios de demanda mediante meta-heurística

Estudiante: Piero do Pazo

Tutor: Carlos Testuri

Informe final de Proyecto de Grado

Instituto de Computación

Facultad de Ingeniería

Universidad de la República

Montevideo, Uruguay

Marzo 2009

Resumen

No es difícil notar en los últimos años, el rápido avance de las redes en telecomunicaciones: internet, sistemas de televisión por cable, etc. llevan cada vez más a la necesidad de redes con un mayor nivel de complejidad, gran cantidad de restricciones, y tamaños enormes.

El proyecto buscó realizar la investigación de meta-heurísticas, en particular de algoritmos genéticos, para aplicar sobre la resolución de un problema de diseño topológico de redes, sometidas a diferentes escenarios. Por esto último podemos entender el caso de tener enormes redes de telecomunicación, por las que viajan flujos de múltiples “mercancías” como demandas a satisfacer entre pares de distintos puntos físicos, compartiéndose a la vez los recursos de la red. Cada escenario es independiente de los otros, y cuenta con una probabilidad dada de acontecer; los flujos de demanda a satisfacer para cada requerimiento, serán diferentes en cada uno de los escenarios. Lo que se busca en el problema que aquí atañe, es el diseño de una sub-red de costo fijo mínimo de instalación, obtenida a partir de la red original, y minimizar la esperanza de los costos operativos por unidad de uso de los arcos, ponderándose la posible realización de todos los escenarios.

Como meta-heurística escogida, se optó por un algoritmo genético (AG), debido a la efectividad y flexibilidad que estos tienen, con la ayuda de otros algoritmos y heurísticas complementarias, para mantener las restricciones que no pueden controlarse con él, como por ejemplo un algoritmo ávido de adición de arcos para incrementar el nivel de confiabilidad de la red, u otros para la búsqueda de caminos arista disjuntos de costo mínimo y el ruteo de demandas. Se realizó un diseño para el funcionamiento del mismo, que incluyó la codificación para su cromosoma, los operadores de selección y reproducción y estrategias de sustitución de población, para dar lugar después a la implementación en lenguaje C++. El resultado fue una aplicación capaz de retornar buenas soluciones en tiempos relativamente cortos. La misma fue probada para 25 problemas, con redes iniciales de grandes cantidades de arcos y demandas. Los resultados se compararon con los obtenidos por otros algoritmos, como forma de medida de que tan buena fue la elección del AG para la resolución de éste tipo de problemas; llegándose a la conclusión de que para aquellos extremadamente grandes, de muchos nodos, arcos y demandas, el AG desarrollado puede devolver buenos resultados en corto tiempo (no más de 5 minutos promedio), siendo en algunos casos, inclusive mejores las soluciones que las obtenidas con un solver basado en Programación Entera Mixta, luego de una hora de ejecución.

Palabras clave: Redes de telecomunicaciones, diseño topológico, confiabilidad, meta-heurísticas, algoritmos genéticos, algoritmos ávidos, programación entera

Tabla de contenidos

Tabla de contenidos 5
1 Introducción 7
1.1 Definición del problema 7
1.1.1 Modelo Algebraico del problema 9
1.2 Motivación 10
1.3 Resultados obtenidos 10
1.4 Conclusiones 11
1.5 Organización del documento 12
2 Algoritmos Genéticos aplicados al diseño de redes 15
2.1 Método de King-Tim et al. 17
2.2 Método de Dengiz et al. 18
2.3 Método de Cox et al. 20
3 Diseño del AG 23
3.1 Codificación del Cromosoma 26
3.1.1 Métodos de conversión 27
3.2 Población inicial 28
3.3 Función de Fitness 29
3.3.1 Función 29
3.4 Algoritmo Genético 29
3.5 Selección 30
3.6 Operadores de reproducción 30
3.6.1 Cruzamiento 31
3.6.2 Mutación 31
3.7 Metodologías para manejo de las restricciones 31
3.7.1 Algoritmo de reparación para 2 arista-conectividad 31
3.7.2 Algoritmo voraz para caminos arista disjuntos 33
3.7.3 Heurística para el ruteo de flujo en la red 35
3.8 Reemplazo de la población 38
3.9 Condición de parada 38
4 Decisiones de Implementación 39
4.1 Librerías 39
4.1.1 MT19937 39

4.1.2 Random 39
4.1.3 Distancia 39
4.1.4 Serializer 40
4.2 TADs 42
4.2.1 Vectores 42
4.2.2 Vector Población 42
4.2.3 Vector de Caminos 43
4.2.4 Cromosoma 44
4.2.5 Grafo 44
4.3 Módulos 45
4.3.1 R2Conectividad 46
4.3.2 Gafodilp 46
4.3.3 FlowRouter 47
4.3.4 Factibilidad 48
4.3.5 Inicializador 48
4.3.6 Metaheurística 49
4.3.7 Problema 51
4.4 Diagrama de Módulos 52
5 Experimentos computacionales 53
5.1 Problemas de prueba 53
5.2 Parámetros para el Algoritmo Genético 55
5.3 Resultados del Algoritmo Genético 56
6 Conclusiones y trabajo futuro 63
Referencias 65
Anexo - Manual de Usuario para el Ejecutable 67

1 Introducción

1.1 Definición del problema

El objetivo es cumplir requisitos de telecomunicación, estableciendo para ello, el diseño inicial y la operación a costo mínimo de una red de soporte. La red se puede modelar como un grafo simple, en el que se tienen posibles nodos, algunos de los que modelan los puntos de origen y destino de las comunicaciones, y enlaces entre nodos, los que modelan canales de telecomunicación, a elegir en el proceso de diseño.

El problema consiste en el diseño de una red, formada por subconjuntos de nodos y aristas de una red potencial, y en determinar los flujos de información (en la forma de múltiples mercancías) de cada requerimiento, con par de nodos origen y destino asociados, de forma de satisfacer la demanda de estos, mientras se minimizan los costos de selección de las aristas y del flujo sobre las mismas. También se agrega a esto, lo que se llaman escenarios: cada escenario tiene para un mismo conjunto de requerimientos de comunicación, una diferente demanda a satisfacer para cada uno de estos últimos. A estos escenarios cabe agregar un factor de aleatoriedad, asociando una probabilidad de ocurrencia a cada uno de estos.

El problema fue planteado por Olivera y otros (Olivera, Robledo, & Testuri, December 15-17, 2008), desarrollando estos un algoritmo basado en GRASP (Greedy Randomized Adaptive Search Procedure) para resolver aproximadamente el problema.

Como ya se mencionó, la red se modelará como un grafo simple no dirigido $G = (N, A)$, con un conjunto de nodos N y otro de arcos entre pares de nodos $A = \{(i,j) \mid i \in N, j \in N\}$. Cada requerimiento k tiene asociado un nodo origen $o(k)$, otro destino $d(k)$ y una demanda d^{ks} a establecer entre ambos, según escenario $s \in S$. A todo esto, se agrega para los requerimientos un nivel de incertidumbre en las demandas. De aquí la existencia de un conjunto de distintos escenarios, con una probabilidad de ocurrencia dada, en donde para cada uno de estos y cada requerimiento, se tiene un nivel de demanda diferente. Uno solo de los escenarios aleatoriamente ocurrirá, por lo que se busca diseñar el horizonte más amplio estratégicamente, para una misma red, la cual podría ser sometida a alguno de estos escenarios.

La instalación de un arco (i,j) trae consigo un costo fijo (por ejemplo: el costo de colocar subterráneamente cables con fibra óptica) y un costo operativo por unidad de flujo enviada desde i hasta j a través del mismo. Además, se establece una capacidad máxima de flujo para cada arco.

A todo esto también cabe agregar las siguientes restricciones:

1. **Balance del flujo**: en cada nodo, el aumento de flujo entrante debe ser igual al flujo saliente, por medio de los arcos que es extremo, a menos que el nodo sea origen u destino para un requerimiento en particular, en cuyo caso tendrá un balance de valor positivo o negativo (dependiendo si el nodo es origen o destino respectivamente), igual al de la demanda a satisfacer entre ambos

2. **Control de capacidad**: en cada arco, la suma de los diferentes flujos pertenecientes a cada demanda que pasan por él no puede superar su capacidad.
3. **Confiabilidad**: debido a que son de ocurrencia bastante común las fallas en los arcos de una red, un método aceptable de diseño debe proveer suficiente redundancia para la supervivencia a estas fallas. Por otro lado, cualquier redundancia incrementa el costo de la red; por esto, para balancear estos conflictos, un mínimo de 2 caminos arista-disjuntos se exigen para todo requerimiento. Se define un valor real pequeño, que se representará como ε de aquí en más, que sirve como medida para determinar un flujo alternativo εd para el ruteo de un flujo de valor *d,* correspondiente a un requerimiento dado.
4. **Control de capacidad máxima y nivel de requerimiento**: para cada arco y requerimiento, el flujo que pase a través del primero, no puede superar su capacidad máxima o una fracción determinada de la demanda correspondiente al requerimiento, según cuál de estos dos valores sea menor. Se mencionó la fracción de una demanda, debido a la restricción de confiabilidad, que no permite que por un arco, que forme parte de un camino entre los nodos origen y destino del requerimiento, pueda llegar a viajar el flujo total para una demanda.

Este problema busca lograr diseñar una red que permita atender varios casos de demandas diferentes, para un mismo conjunto de requerimientos, manteniendo las restricciones de capacidad en las aristas y requerimientos de confiabilidad que se deben de llevar a cabo.

La minimización del costo de las conexiones y de la esperanza de los costos operativos para todos los escenarios, junto a las restricciones de confiabilidad en la red, pueden llegarse a ver como interrelacionadas entre sí. La minimización del costo fijo de instalación de los arcos se contrapone con el objetivo de maximizar las propiedades de confiabilidad de la red, así como también un cambio en la asignación del flujo traerá distintos costos operativos, al viajar éste por diferentes caminos. La Figura 1 muestra como estos sub-problemas están relacionados. Las flechas indican la dirección de incidencia. De todos los factores que afectan el costo, solamente la topología no se ve influida por los otros.

Figura 1: Sub-problemas del problema de diseño y sus interrelaciones

1.1.1 Modelo Algebraico del problema

Sea A^p el conjunto de arcos posibles de un grafo dirigido equivalente en el que se establecen ambas direcciones para cada arco: $A^p = \{(i,j) \mid (i,j) \in A \text{ o } (j,i) \in A\}$.

Cada requerimiento $k \in K$ tiene asociado un nodo origen $o(k) \in N$ y un nodo destino $d(k) \in N$. También se cuenta con un conjunto discreto de escenarios de demanda S, en donde para cada escenario $s \in S$ y requerimiento k se tiene una demanda d^{ks}, a enviar desde $o(k)$ hacia $d(k)$. Los escenarios tienen una probabilidad de ocurrencia asociada, p_s, tal que $\sum_{s \in S} p_{s=1}$.

La instalación del arco (i,j) tiene un costo fijo f_{ij}, y su uso un costo operativo c_{ij} por unidad de flujo. Además se establece una capacidad máxima de flujo para cada arco, u_{ij}.

Se definen las variables:

- x_{ij}, binaria, que indica si se utiliza el arco (i,j)
- y_{ij}^{ks}, real, que indica la demanda del requerimiento k en el arco (i,j) para el escenario s.

La formulación algebraica del problema es:

$$min_{x,y} \sum_{(i,j)\in A} f_{ij} + \sum_{s\in S}\sum_{k\in K}\sum_{(i,j)\in A} p_s c_{ij}(y_{ij}^{ks} + y_{ji}^{ks}) \qquad (1)$$

$$s.a. \sum_{j:(i,j)\in A^p} y_{ij}^{ks} - \sum_{j:(j,i)\in A^p} y_{ji}^{ks} = b_i^{ks}, \qquad \forall i \in N, \forall k \in K, \forall s \in S, \qquad (2)$$

$$\sum_{k\in K} (y_{ij}^{ks} + y_{ji}^{ks}) \le u_{ij} x_{ij}, \qquad \forall (i,j) \in A, \forall s \in S, \qquad (3)$$

$$y_{ij}^{ks} + y_{ji}^{ks} \le g_{ij}^{ks} x_{ij} \qquad \forall (i,j) \in A, \forall k \in K, \forall s \in S, \qquad (4)$$

$$y_{ij}^{ks} \ge 0, \qquad \forall (i,j) \in A^p, \forall k \in K, \forall s \in S,$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A,$$

donde

$$g_{ij}^{ks} := min\{u_{ij}, (1-\varepsilon)d^{ks}\} \; y \; b_i^{ks} := \begin{cases} d^{ks} \; si \; i = o(k), \\ -d^{ks} \; si \; i = d(k), \\ 0 \; en \; otro \; caso \end{cases} \qquad (5)$$

La función objetivo está modelada por (1). La restricción de balance de flujo está representada en (2), (3) la activación de arcos y control de capacidad, y (4) control de capacidad máxima, confiabilidad y nivel de demanda para cada requerimiento; la confiabilidad se lleva a cabo imponiendo un pequeño incremento del flujo ($\varepsilon > 0$) , para que cada demanda siga un camino alternativo.

1.2 Motivación

La búsqueda de soluciones a este tipo de problemas, hoy en día resulta cada vez más interesante de tratar, partiendo de la diversidad de nuevas modalidades que se están dando en las telecomunicaciones (redes telefónicas preferentemente, sistemas de televisión por cable, etc.). Se espera que estas sean confiables, manteniéndose rutas alternativas en caso de rotura de arcos, que no tengan una carga alta de flujo de información, ya que esto llevaría a enlentecer la comunicación necesitándose ocupar más ancho de banda y que los costos fijos de instalación (por ejemplo para una línea de fibra óptica subterránea) sean los mínimos. Esto nos trae aquí, al planteamiento de nuestro problema y la búsqueda de una metodología que brinde soluciones satisfactorias para él, en tiempos no excesivamente largos.

Un problema de diseño de redes como el presentado, es un problema de optimización combinatorio del tipo NP-difícil, donde el espacio de búsqueda para un grafo completo con un conjunto de nodos N y A arcos posibles es: $A^{|N|(|N|-1)/2}$ (Smith & Dengiz, 2000). Los tradicionales algoritmos de optimización, tales como programación lineal y métodos de programación entera (como programación entera mixta) tendrán tiempos de ejecución exponencial a medida que el tamaño del problema aumenta. Para esto, se buscan meta-heurísticas alternativas que permitan encontrar buenas soluciones en tiempos razonables. Varias de estas se han aplicado al diseño de redes de telecomunicaciones confiables, al momento de quererse resolver problemas de dimensiones reales en tiempos relativamente razonables.

1.3 Resultados obtenidos

Se optó por trabajar con algoritmos genéticos como meta-heurística a utilizar, aprovechando las ventajas que estos pueden brindar para la obtención de soluciones factibles, ante problemas como el planteado para éste proyecto; ventajas como: exploración de varias soluciones en paralelo o mayor flexibilidad para cualquier problema que se busque resolver, entre otras.

Se diseñó el algoritmo genético (AG) adaptándolo para la búsqueda de soluciones factibles al problema. El mismo contó con el esquema básico de un AG, mientras que el cromosoma consistió en una tira de genes binarios, de tamaño igual a $n(n-1)/2$, siendo n la cantidad total de nodos del grafo de partida del problema, indicando la presencia o no de un arco en la solución, por la existencia de un 1 en el gen correspondiente a éste en el cromosoma. Como operador de selección, se implementó una variación de la selección por ruleta, dando mayor probabilidad de ser escogidos para reproducción a aquellos individuos de menor valor objetivo. Los operadores de reproducción fueron los clásicos, gracias al diseño del cromosoma escogido: cruzamiento en dos puntos y mutación de un bit, con mayor probabilidad de ocurrencia para el primero. El reemplazo en la población por nuevos individuos, consistió en una estrategia de reemplazo de los individuos menos adaptados.

Por otra parte, para la verificación de las restricciones que atañen al problema, se manejaron otros algoritmos complementarios, a los que fueron sometidos cada nuevo candidato a nueva solución factible dentro de la población del AG El primero es un algoritmo de reparación para lograr que el grafo cumpla la propiedad de 2 arista-conectividad[1], propiedad típica de redes altamente confiables. Una vez aplicado éste algoritmo de reparación, se utiliza otro para la búsqueda de más de dos caminos arista disjuntos, asegurando así la restricción de confiabilidad del problema. Se trata de un algoritmo ávido que busca caminos de menor costo (por medio del algoritmo de Dijkstra) en sub-grafos resultantes del original, de los cuales se van retirando las aristas de cada camino hallado. Finalmente, para las restricciones de control de capacidad, nivel de requerimiento y balance del flujo, se utilizó una heurística encargada del enrutamiento de los flujos por los caminos encontrados en el algoritmo previamente mencionado.

La implementación se realizó en lenguaje C++, realizándose diferentes módulos responsables de cada uno de los algoritmos diseñados (ej: un módulo encargado del AG, otro para encontrar los caminos arista disjuntos, etc.), así como las estructuras necesarias para la representación de los diversos elementos manejados (grafos, cromosomas, listas de caminos, etc.). Se obtuvo un programa que recibe la configuración del problema (nodos y arcos del grafo, demandas y escenarios) por medio de un archivo de entrada, y los parámetros que son ingresados por pantalla (tamaño de la población, número de generaciones), para luego de finalizar su ejecución imprimir los resultados en más de un archivo de salida.

Veinticinco problemas fueron sometidos como prueba al AG implementado. Estos presentaron diferentes tamaños en el número de nodos y arcos, así como también en la cantidad de requerimientos de comunicación y escenarios; por otra parte, algunos de ellos contaban con capacidades máximas en los arcos más holgadas que otros, los cuales fueron considerados como más “pesados”. Los resultados obtenidos fueron comparados con los de un solver basado en Mixed Integer Programming (MIP) y también con los del algoritmo GRASP desarrollado por Olivera y otros. Para los primeros problemas testeados (los de menor cantidad de nodos, arcos, demandas y escenarios) se encontraron buenas soluciones, con una distancia no mayor del 11% al óptimo, incluso en un caso se halló el propio óptimo global. Mientras que para aquellos de mayor cantidad de componentes para el problema, se obtuvieron mejores resultados en la mayoría de ellos, llegándose a encontrar mejores soluciones casi hasta un 40 y 7% que las logradas por MIP y GRASP respectivamente. Sin embargo, en dos de los problemas considerados como “pesados”, los resultados no fueron del todo buenos, dejando uno sin solución alguna encontrada y otro con una distancia del 15% respecto al óptimo.

1.4 Conclusiones

Problemas de optimización para diseño topológico de redes, como el presentado para éste proyecto, con restricciones no triviales de confiabilidad y *multi-commodity network flow*, están catalogados como del tipo NP-Dificil, por lo que se atacó una investigación de

[1] Un grafo presenta 2 arista-conectividad si todos sus nodos poseen grado mayor o igual a dos

su resolución por medio de una meta-heurística. Fueron escogidos los algoritmos genéticos, desarrollándose una aplicación basada en los mismos, la cual retorna soluciones de forma rápida y efectiva, bastantes próximas a las óptimas, si no resultan en la óptima misma. Pudo constatarse, viendo los costos de las soluciones resultantes de las pruebas a las que se sometió, que a medida que el tamaño de la cantidad de nodos y aristas de la red aumentan, se obtienen resultados mejores que para con otros algoritmos, en tiempos más cortos. Por otra parte, para algunos ejemplos de problemas con capacidad acotada en los arcos y mayores valores de demanda para los escenarios, la eficiencia del algoritmo genético merma, aunque no sin dejar de dar resultados para la mayoría, necesitándose seguramente de más trabajo (más pruebas, nuevas configuraciones) para lograr mejores resultados.

1.5 Organización del documento

El resto del documento estará dividido en 5 capítulos más, los cuales contendrán cada uno sus correspondientes sub-secciones.

El capítulo 2 presenta la meta-heurística escogida para la resolución del problema planteado, las razones de su elección, así como también un estudio del arte en cuanto a su uso en problemas de similar naturaleza, cómo se adaptó la misma y la conveniencia o no de las características de cada uno para el caso aquí planteado.

El capítulo 3 mostrará el diseño de la metodología escogida. Las primeras sub-secciones expondrán aspectos importantes para la realización de la meta-heurística, conteniendo decisiones importantes con los que se desarrollará la ejecución de ésta, mientras que las restantes mostrarán aquellos algoritmos y heurísticas auxiliares escogidos y modificados para el cubrimiento de funcionalidades que no pueden satisfacerse por parte de la meta-heurística.

El capítulo 4 enseña las decisiones fundamentales que se llevaron a cabo al momento de la implementación, mostrando en sub-secciones los módulos diseñados y su funcionamiento, diferenciando de éstos aquellos que son tipos de datos de los que contienen la parte fundamental de la lógica de la metodología. También se dará un esquema de la relación entre estos módulos.

El capítulo 5 está dividido en dos partes: la primera da una muestra de los diferentes tipos de problemas que serán puestos a modo de investigación por parte de la meta-heurística, qué contienen cada uno, qué se busca evaluar con ellos y los parámetros de configuración especialmente usados en la búsqueda de soluciones óptimas a estos. La segunda presenta los resultados obtenidos y un análisis de estos.

El capítulo 6 mostrará las conclusiones obtenidas no solamente por los resultados logrados en el capítulo 5, sino también por lo analizado y desarrollado para el problema general planteado. También dará las consideraciones concernientes respecto al trabajo futuro a realizar para con el proyecto.

Sobre el final, además de la bibliografía recomendada para una mejor comprensión de éste documento, se agrega un breve manual para el uso del programa ejecutable resultante de la implementación de la meta-heurística.

2 Algoritmos Genéticos aplicados al diseño de redes

El problema presentado para este proyecto en particular, así como sus varias versiones, han sido estudiados de forma muy variada en la literatura, dejando lugar a una larga lista de métodos disponibles. Se ha desarrollado un universo amplio de diversas heurísticas: heurísitcas ávidas, branch and bound, simulated annealing, algoritmos genéticos, búsqueda tabú, son algunos de los tantos ejemplos. La mayoría de los problemas relacionados con diseños de redes y resueltos con las mismas, siempre presentan una combinación de las siguientes características en común: diseño de una topología de costo mínimo, ruteo de flujo, confiabilidad, asignación de capacidad.

Los algoritmos genéticos (Gil Londoño, 2006) se muestran como buenos candidatos, permitiendo una mayor flexibilidad y robustez. Estos presentan la ventaja de poder explorar espacios de soluciones en múltiples direcciones a la vez, teniendo así un comportamiento paralelo, mientras que la mayoría de los otros algoritmos son en serie y sólo pueden explorar el espacio de soluciones dirigiéndose hacia una solución al mismo tiempo. Por ésta misma característica, realizar una exploración exhaustiva del espacio se hace prácticamente imposible, con inconvenientes de caídas en óptimos locales (soluciones que son mejores que todas las similares a ella, pero que no son mejores que otras distintas situadas en algún otro lugar del espacio de soluciones); el algoritmo genético presenta la ventaja de escapar a los óptimos locales y posiblemente lograr descubrir el global, debido al cruzamiento entre integrantes de la población, ya que sin éste operador de reproducción, cada solución individual iría por su cuenta, explorando el espacio de búsqueda en sus inmediaciones, sin referencia alguna de lo que el resto de los individuos pudo haber descubierto. Aunque, por otro lado, el cruzamiento no garantiza totalmente dicho escape a óptimos locales, por lo que es esencial la ayuda de la mutación de individuos, ya que pueden estarse realizando apareamientos entre individuos cercanos en el espacio de búsqueda, que dejaran nuevas soluciones muy similares a sus progenitores. Otros procesos distintos de los algoritmos genéticos, pueden llegar a aplicar una estrategia para resolver problemas, que puede llegar a resultar satisfactoria y encontrar óptimos locales; sin embargo, si estos continúan utilizando la misma por considerarla buena, excluyendo las demás, pueden estar descartando mejores estrategias que existan. Otra ventaja que hace los AG interesantes de emplear para este problema, es que aunque estos no devuelvan una solución óptima al problema, casi siempre pueden devolver al menos una muy buena. Esto se debe a que inicialmente, el AG genera una población inicial aleatoriamente diversa sobre el espacio de soluciones, llevando a que pequeñas mutaciones permitan a cada individuo explorar sus proximidades, mientras que la selección orienta el progreso, llevando a los descendientes del algoritmo hacia mejores zonas del espacio de búsqueda. Como desventaja puede mencionarse la convergencia prematura que alcance darse en éstos. Si se tiene un individuo muy apto por encima del promedio de sus competidores, emerge muy pronto durante la ejecución y llegará a ser aquel con el que preferentemente escojan reproducirse el resto de los individuos, lográndose mermar la diversidad de la población de forma rápida, provocando una convergencia hacia el óptimo local en ese espacio de soluciones.

Varias aplicaciones cuentan con el uso de algoritmos genéticos para el diseño de redes; por citar algunos ejemplos, se tiene lo desarrollado por Kumar y otros (Kumar, Pathak, Gupta, & Parsaei, 1995), considerando el diámetro máximo de la red (mínimo número de arcos entre cualquier par de nodos) como una restricción y calculando la confiabilidad de todos los nodos terminales (subconjunto de nodos que indefectiblemente deben estar presentes en la solución). Abuali y otros (Abuali, Schoenefeld, & Wainwright, 1994) asignaron todos los nodos terminales a sitios concentrados, buscando así minimizar costos, mientras se consideran las capacidades usando un AG, mas sin suponer confiabilidad. También se han aplicado algoritmos genéticos híbridos para la resolución del Problema Generalizado de Steiner (Nesmachnow, Cancela, & Alba, 2004). La idea detrás de estos AG híbridos se basa en la inclusión de un método de búsqueda de conocimiento dependiente del problema para mejorar el mecanismo de búsqueda, pudiéndose aplicar a nivel de la codificación así como en operadores específicos o combinando técnicas. Nesmachnow y otros utilizan esta última, haciendo una combinación de los algoritmos genéticos con simulated annealing.

Al momento de implementarse un AG, de las primeras consideraciones a tener en cuenta es la representación del problema por medio de él, esto se entiende como la codificación del cromosoma y sus alelos de manera adecuada. Esta representación deberá de ser robusta, evitando cambios aleatorios que traigan resultados erróneos. Por otra parte, es fundamental la correcta elección de una función objetivo como medida de adaptación de la solución al problema (función de fitness). En caso de escogerla mal o definirla de forma incorrecta, se podrá tener un AG que resuelva el problema equivocado, o simplemente no lo resuelva.

Una vez escogidos los algoritmos genéticos como meta-heurística, se estudiaron diversas variaciones de los mismos para ser aplicados en problemas de diseño de redes, similares al que nos atañe. Entre la bibliografía consultada se encontró más de un caso, donde se utiliza un AG de forma clásica prácticamente, adhiriendo algoritmos o meta-heurísticas adicionales para el manejo de restricciones que no puedan cubrirse con el primero. Nesmachnow y otros aplican una heurística para descartar soluciones cuyos nodos terminales no tengan grado superior al máximo requerimiento de conexión para el mismo. En caso de ser factibles, se aplica un algoritmo para hallar caminos entre pares de nodos terminales (origen y destino), los cuales, en caso de coincidir con el flujo máximo entre origen y destino, estarán determinando que la solución sea factible. Blessing, al momento de plantear el algoritmo genético, define un cromosoma para la representación de la topología, pero para las demás restricciones a las que se impone el problema se presentan otras metodologías adicionales, una de ellas será mencionada más adelante (sección 2.1 de éste capítulo).

Teniendo en cuenta lo mencionado anteriormente, e investigando más profundamente sobre esta idea, caben destacar tres métodos, muy diferentes entre sí, que se hallaron interesantes y más cercanos a la realidad a resolver. Se explicarán junto con sus ventajas y desventajas, identificando por el nombre de sus autores.

2.1 Método de King-Tim et al.

El método propuesto por King-Tim y otros (Ko, Tang, Chan, Man, & Kwong, 1997) se destaca por utilizar un cromosoma para el diseño topológico de la red y otro separado para el ruteo del flujo a enviar entre dos nodos *i,j* con demanda *d*, resolviendo cada problema por medio de un AG diferente. Como puede verse en la Figura 2, existe además una rutina de optimización, que cuenta con un proceso de corrección para el caso en que un miembro de la población no cumpla con la restricción de bi-conectividad, basado en un algoritmo de depth-first search; en caso contrario, se realiza la optimización del ruteo, utilizando otro AG, para finalizar evaluando el fitness de la nueva solución lograda.

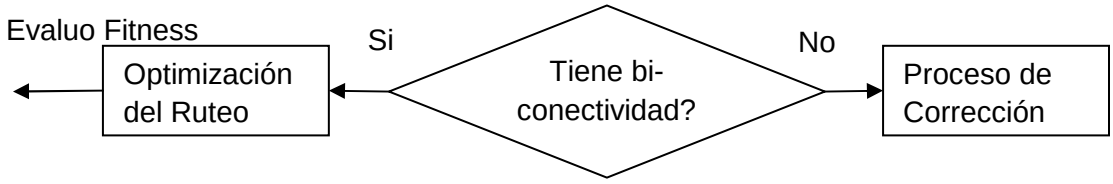


Figura 2: Rutina de optimización

Topología

El cromosoma principal para el diseño topológico de la red, es un *string* de valores binarios, de tamaño $n(n\text{-}1)/2$, siendo *n* el número de nodos del grafo. Este tamaño es el de la cantidad de arcos de un grafo completo de *n* nodos. La presencia o no de un arco en la solución estará representado por el valor de los alelos 1 o 0 respectivamente, en el gen correspondiente al nodo que se hace referencia.

Ejemplo:

Para un grafo de 5 nodos, etiquetados como 1,2,3,4,5, se puede tener una representación del cromosoma como en el grafo de la Figura 3.

Ruteo

Para el problema del ruteo y flujo en la red, se considera el uso de otro cromosoma, que consiste en una lista de caminos $P = \{p_1, p_2, \ldots, p_{n(n-1)/2}\}$, que representa el grafo completo. Cada camino p_k es una ruta particular entre los nodos *i* y *j*, donde la posición *k* dentro del cromosoma puede calcularse a partir de los nodos *i, j* de la siguiente manera:

$$k_{ij} = ni + j - \frac{(i+1)(i+2)}{2}$$

Teniendo esta representación de cromosoma, el ruteo se puede realizar asignando a cada uno de estos caminos el flujo correspondiente, para luego evaluar el fitness del esquema de ruteo logrado.

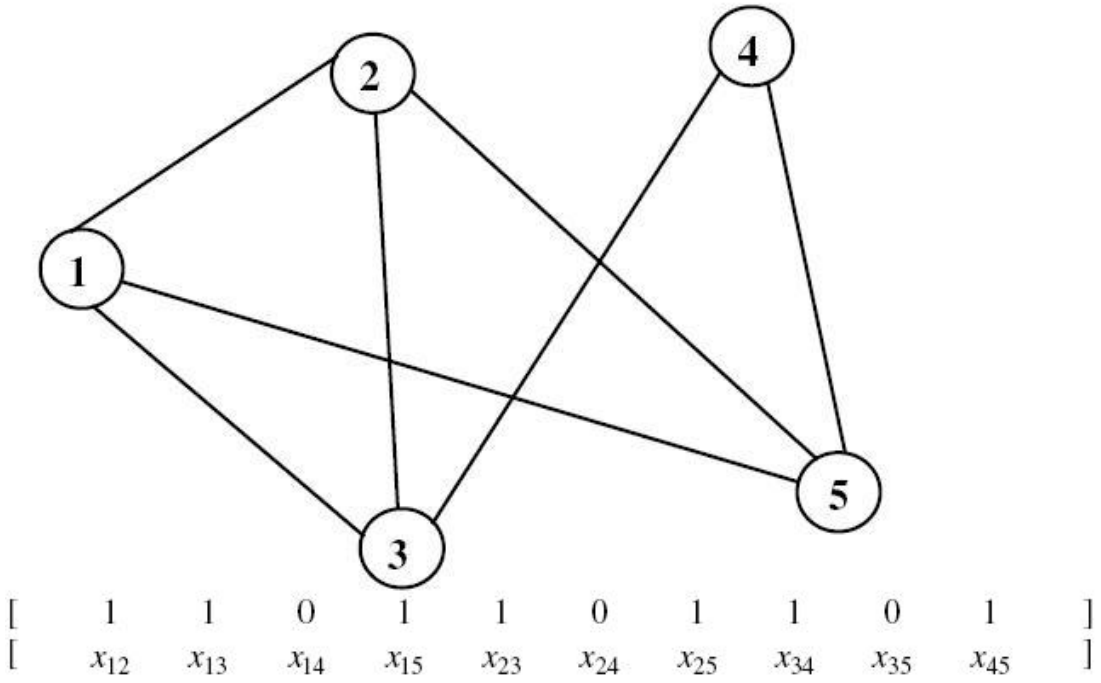


Figura 3: Ejemplo de cromosoma para la topología

Ventajas y desventajas

Ventajas	Desventajas
Resuelve el problema de diseño topológico y enrutamiento (un cromosoma por problema)	El cromosoma de ruteo puede ser difícil de manejar – cada alelo es un camino conformado por una lista de arcos
Cromosoma binario para topología – operadores sencillos de evolución (cruzamiento y mutación)	Redundancia de información en cromosoma *P*: pueden haber sub-caminos dentro de otros
	Un camino por cada par de nodos, lo cual elimina toda confiabilidad
	El cromosoma de topología puede generar soluciones no factibles.

2.2 Método de Dengiz et al.

Este método, propuesto por Dengiz y otros (Dengiz, Altiparmak, & Smith, 1997), se destaca por tener un solo cromosoma para representar la presencia de un arco en la solución. Entre la población se admiten soluciones no factibles que no cumplan la restricción de confiabilidad, agregando un valor de penalización en la función de fitness. Con esta admisión, se permite aprovechar las bondades que estos individuos no factibles puedan tener al realizar un cruzamiento con otros individuos, o al realizar una mutación.

El uso de una función de penalización puede expresarse teóricamente de la siguiente forma:

Si tenemos un problema de optimización

$$\min z(x) \qquad (1)$$

$$\text{s.a. } x \in A$$

Donde x es un vector, y $x \in A$ es una restricción, podemos sustituir el problema (1) por

$$\min z(x) - \pi(d(x,A)) \quad (2)$$

donde $d(x,A)$ es una función métrica que describe la "distancia" del vector x al conjunto A y $\pi(.)$ es una función de penalidad incremental, tal que $\pi(0) = 0$; de esto, puede notarse entonces que cualquier solución óptima para (1) es también una solución óptima para (2), cuando $d(x,A) = 0$.

En muchos casos, soluciones no factibles se encuentran en el espacio de búsqueda más cerca del óptimo global que otras factibles, por esto puede ser conveniente a veces admitir las primeras dentro de la población del algoritmo genético.

Como acotación a esta particularidad del método, es evidente destacar la encrucijada, al momento del diseño de un AG, de 2 decisiones fundamentales:

1. Existencia de soluciones no factibles dentro de la población, agregando un costo de penalidad a la función de fitness.
2. Descarte de soluciones no factibles, evitando con esto la tarea de cuantificar soluciones y de introducir modelos de penalización para el fitness.

Además de la implementación del algoritmo genético también se realizan otros algoritmos para verificar la conectividad del grafo, así como también para modificarlo en caso de que la solución no presente 2 arista-conectividad. A continuación se enumeran las verificaciones realizadas para cada nuevo individuo de la población:

1. Chequeo de conectividad para un árbol de expansión en cada nueva red
2. Para los que pasen el chequeo anterior, se verifica que el grafo presente 2 arista-conectividad para todo nodo, aplicando un algoritmo ávido de adición de arcos.
3. Finalmente, para los que pasen los 2 pasos anteriores, se realiza un cálculo para el límite superior del valor de confiabilidad de la red candidata, $R_U(x)$.

Aquellas redes que cumplan la restricción $R(x) \geq R_0$ donde $R(x)$ es la confiabilidad de la red y R_0 es el mínimo requerimiento de confiabilidad, se considerarán como factibles.

Codificación del cromosoma

A cada arco se le asigna un valor entero (yendo desde 1 hasta la cantidad máxima de arcos existentes), y la presencia de ese arco en la solución es indicada por la presencia de su entero correspondiente dentro del cromosoma, representándose éste último como un *string* de valores enteros de largo variable.

Ejemplo:

Una representación del cromosoma para el grafo de la Figura 4 sería de la forma: [1 4 5 6 9 11 12 13 14 15]

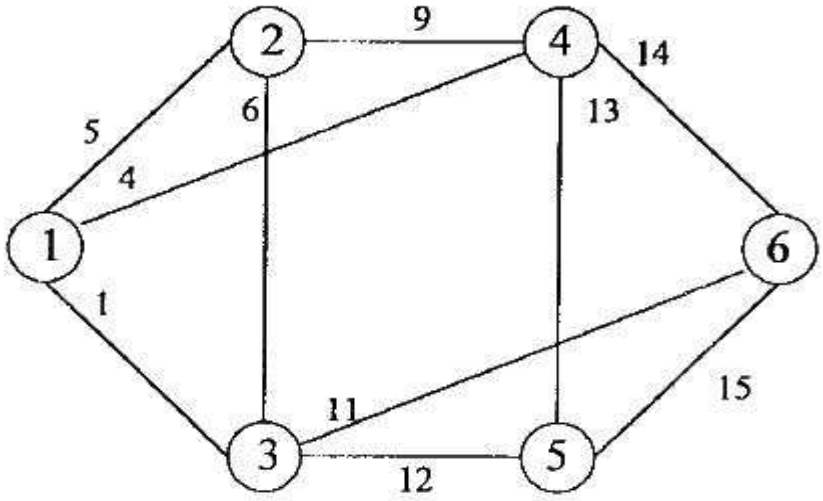


Figura 4

Ventajas y desventajas

Ventajas	Desventajas
Maneja un solo cromosoma	No presenta ningún método para el cumplimiento de requerimientos entre pares de nodos (multi-commodity network flow)
Cromosoma de largo variable	Operadores de cruzamiento y mutación no son aplicados directamente sobre el cromosoma, sino sobre el grafo que estos representan.
Algoritmo ávido para asegurar la bi-conectividad	

2.3 Método de Cox et al.

La idea básica del método propuesto por Cox y otros (Cox, Orvosh, Davis, & Qiu, 1993) consiste en considerar las capacidades de los arcos y realizar un re-ruteo en caso de falla de los mismos, utilizando un AG adaptado.

Se tiene un conjunto ***X*** de capacidades de las aristas del grafo, sujeto a 3 consideraciones:

1. **Restricción de ruteo del flujo**: para cada requerimiento *w* entre un par de nodos, con demanda d_w, debe existir un conjunto caminos p_w con capacidad suficiente para transportar la demanda.
2. **Restricción de confiabilidad**: dado un nivel de confiabilidad $0 \leq \varepsilon \leq 1$, para cada requerimiento *w*, con demanda d_w, un aumento de flujo $\varepsilon * d_w$ debe de poderse

"rutear" usando las capacidades de ***X****, donde ***X**** es como ***X*** excepto que un elemento de ***X*** es establecido en 0.

3. **Costo objetivo**: consiste en la minimización del costo de los arcos de la red.

Para los 2 primeros puntos se utilizan algoritmos que se encargan de mantener el cumplimiento de las restricciones, mientras que el tercero es evaluado por medio de una función de costo que considera una variable de suma de penalidades, para el caso de graves violaciones de restricciones durante el proceso de evaluación, más el costo fijo de cada arco y el costo variable por unidad de flujo que atraviese el arco.

Cromosoma

Sea ***W*** una matriz de tamaño *nxn* conteniendo las demandas entre todo par de nodos. El cromosoma está compuesto de tres partes ***XW'W''***, donde ***X*** es un conjunto de capacidades de arcos, y ***W'*** y ***W''*** son permutaciones aleatorias de la matriz ***W***.

Al inicio del AG, se utiliza un algoritmo de Dijkstra para obtener un grupo de caminos, para cada requerimiento, los cuales serán ordenados por costo en orden ascendente, para luego aplicar la evaluación del cromosoma y el procedimiento de reparación para que se sigan cumpliendo los requerimientos de confiabilidad. La evaluación del cromosoma se realiza considerando el primer componente ***X***, evaluado con respecto a la restricción de ruteo del flujo usando el segundo componente ***W'***. Luego es evaluado con respecto a la restricción de confiabilidad, usando el tercer componente ***W''***.

Ventajas y desventajas

Ventajas	Desventajas
Resuelve el problema de diseño topológico de la red, ruteo del flujo y restricción de confiabilidad	Cromosoma de muy difícil manejo
	4 nuevas operaciones para cruzamiento, y mutación, las cuales se aplican aleatoriamente, usando una selección por ruleta para elegir cual usar. En el caso de cruzamiento, se aplica un operador distinto a cada parte del cromosoma
	Maneja un conjunto de caminos para cada requerimiento, paralelamente con el cromosoma, lo que puede tener como consecuencia manejar información redundante, como caminos repetidos entre más de un individuo de la población

3 Diseño del AG

En el capítulo anterior se destacaron tres metodologías donde se aplica un algoritmo genético para la resolución de problemas de diseño de topologías de redes, combinados, aunque no con todos a la vez, con confiabilidad, ruteo del flujo y minimización de costos. Viendo las ventajas y desventajas de cada uno, se puede llegar a la conclusión que así como son presentados, sin alguna modificación, no pueden aplicarse al problema que atañe a éste proyecto. El método de Cox *et al.* parecería ser aquel que reúne la mayor cantidad de características necesarias: tiene una topología, ruteo del flujo y maneja la restricción de supervivencia de la red, sin embargo, la metodología usada por los autores no está muy bien explicada en el documento, así como la codificación del cromosoma es muy complicada de manejar. En el método de Dengiz *et al.* no se da un algoritmo o heurística para el ruteo del flujo, ya que no forma parte del problema por ellos resueltos y en Kin-Tim *et al.* no se tiene en cuenta la restricción de confiabilidad.

Si bien estos métodos no pueden ser aplicados al problema tal como están presentados, surge sí la idea de combinar lo mejor de cada uno, para lograr un diseño efectivo y robusto, logrando que las carencias de uno, sean suplantadas por los beneficios del otro. Está claro que no se podrá resolver el problema aplicando un algoritmo genético en su forma simple, con un único cromosoma codificado de manera de manejar todas las restricciones, ya que esto llevaría a un diseño del mismo muy complejo y a un manejo poco común. Por otra parte está el comportamiento del AG, ya que requeriría chequeos de factibilidad, procedimientos para control de capacidad, etc. Un diseño para un AG que se encargue de todas estas particularidades, es delicado de llevar a cabo y que aún mantenga sus ventajas principales, como por ejemplo, que éste no sepa cual problema debe resolver en particular.

Teniendo todo lo expuesto en cuenta, finalmente se optó por experimentar con un algoritmo genético simple. En la Figura 5 se presenta un diagrama de su comportamiento, con las siguientes características:

1. Un único cromosoma con genes binarios, para la representación de la topología.
2. De Dengiz *et al.* se puede aplicar su algoritmo de reparación, en caso que la solución generada no cumpla la restricción de 2 arista-conectividad para sus nodos terminales.
3. Un algoritmo ávido para encontrar conjuntos de caminos arista disjuntos entre pares de nodos origen y destino, acompañado de otro algoritmo para hallar caminos de menor costo. Serán los caminos candidatos por donde enviarse la demanda.
4. De la metodología de Cox *et al*. puede adaptarse su heurística para el ruteo del flujo, agregando una iteración más para cada escenario (por encima de las de para cada requerimiento)
5. Se descartaron las soluciones no factibles, evitando con esto introducir funciones con costos de penalidades sobre el valor de fitness. Se considera como no factible una solución que falle en la verificación de alguno de estos puntos:

a. Existen nodos terminales (origen o destino) con grado de aristas menor que 2. Toda solución que presente esta característica, luego de aplicársele el algoritmo de reparación mencionado en el punto 2, pasará a ser candidata a nueva integrante de la población en caso de superar las verificaciones b y c siguientes.
b. Toda solución que para cada requerimiento no presenta al menos 2 caminos arista disjuntos entre los nodos origen y destino de éste
c. Toda solución para la cual no se pueda realizar el ruteo del flujo para cada escenario.

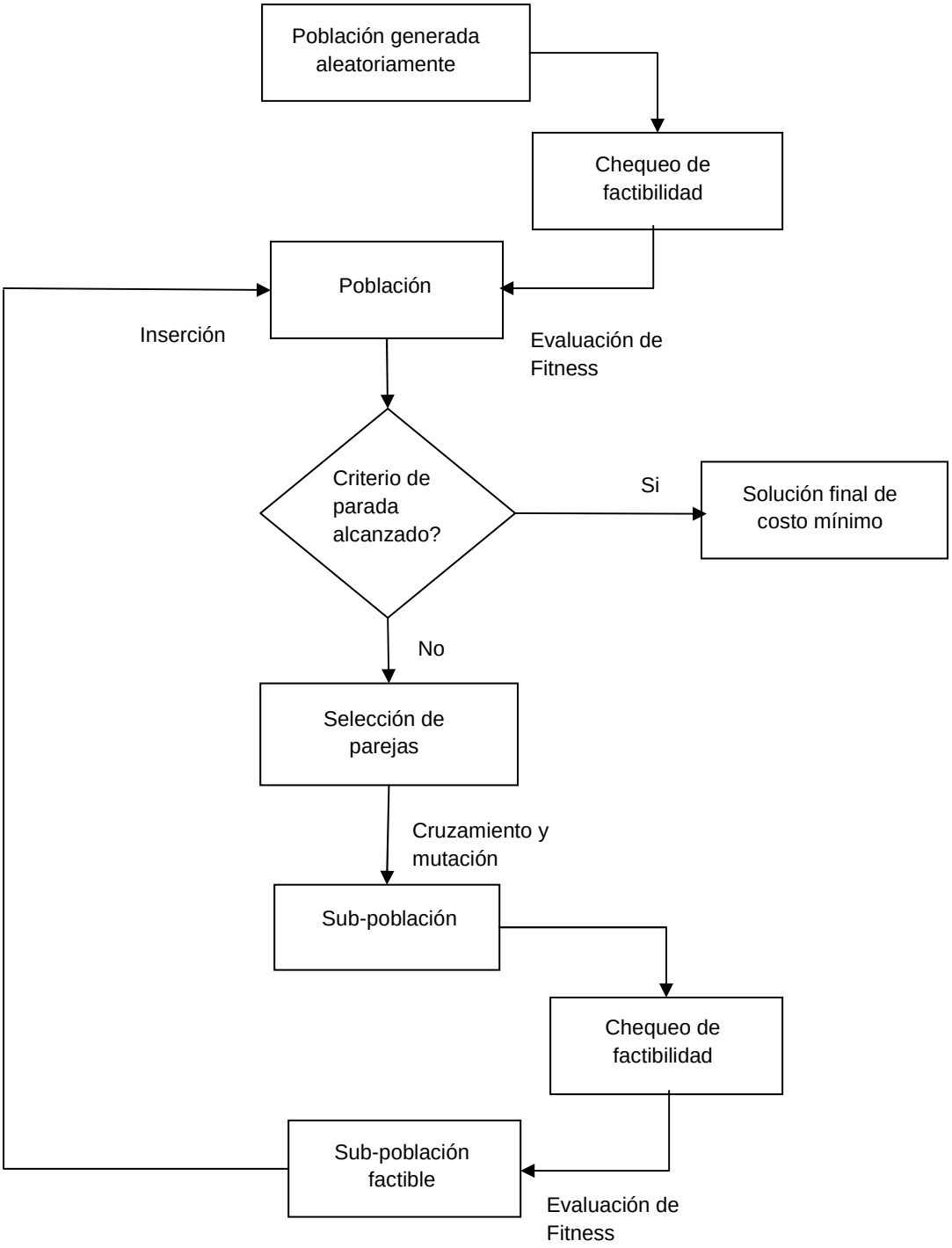


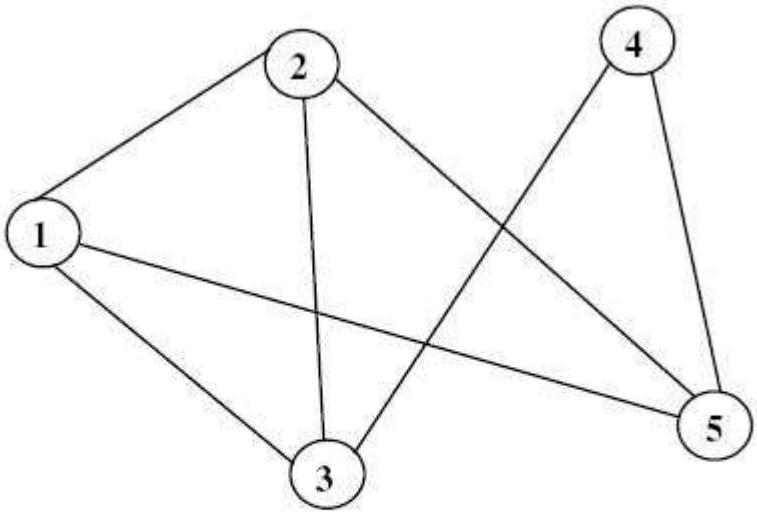
Figura 5: Pasos del algoritmo genético

En caso de no lograr obtenerse un algoritmo genético capaz de encontrar soluciones de buena calidad, en cortos espacios de tiempo, o que simplemente no logre alcanzar ningún óptimo local, desechando todas las soluciones candidatas como no factibles, se replanteará el caso, buscando modificaciones al AG (su cromosoma, operadores de evolución) así como a los algoritmos adicionales.

A continuación se presenta el diseño aplicado al algoritmo genético empleado al problema del diseño de redes de telecomunicaciones con múltiples escenarios de demanda. Además de la descripción de los elementos utilizados para este algoritmo, también se muestran los algoritmos que le complementan.

3.1 Codificación del Cromosoma

El cromosoma para el diseño topológico de la red, es representado con un vector de valores binarios, de tamaño $n(n\text{-}1)/2$, siendo n el número de nodos del grafo. Este tamaño es el de la cantidad de arcos de un grafo completo de n nodos. La presencia o no de un arco en la solución está representada por el valor 1 o 0 (los alelos del cromosoma) respectivamente, en el gen correspondiente a la arista que se hace referencia. Ver ejemplo de codificación en Figura 6.



El cromosoma correspondiente a la red de la figura es

[	1	1	0	1	1	0	1	1	0	1	]
[	x_{12}	x_{13}	x_{14}	x_{15}	x_{23}	x_{24}	x_{25}	x_{34}	x_{35}	x_{45}	]

Figura 6: Ejemplo de codificación del cromosoma

Como el grafo de partida no tiene porque ser un grafo completo, se tuvo un vector adicional al cromosoma, de igual tamaño que este, que guardó la información estática de cuales aristas son admitidas en las soluciones factibles. Llamaremos a este vector *VAA* (Vector de aristas admitidas) y contiene, así como ocurre con el cromosoma, valores binarios, con un 1 indicando que el arco pertenece al grafo de partida y 0 en caso contrario. Las posiciones para cada arista en ambos fueron las mismas.

Para mantener una correlación entre la posición de un valor binario en estos dos vectores y el par de nodos extremos del arco que este representa, se muestra a continuación la metodología de mapeo.

3.1.1 Métodos de conversión

Para lograr la conversión de una posición en el cromosoma al par de nodos que conforman la arista que esta representa, se comienza por considerar una representación del grafo *G*, con el que se esté trabajando, como una matriz de adyacencia de tamaño *nn* (siendo *n* el número de nodos existente), donde un 1 o 0 en la fila *i*, columna *j* representa la existencia o no de un arco respectivamente entre los nodos *i* y *j*. Si el grafo es no dirigido, esta matriz tiene la particularidad de ser simétrica.

Se dan a continuación los métodos para, dados dos nodos *i y j*, retornar la posición en el vector que se corresponde con la posición (*i,j*) de la matriz, e inversamente.

Se define k_{ij} como un natural que representa la posición dentro del vector cromosoma, correspondiente a la arista (*i*,*j*)

Usando el cromosoma siguiente de ejemplo (para un grafo de 5 nodos):

[1, 1, 0, 1, 1, 0, 1, 1, 0, 1]

k_{01}=0, k_{02}=1,…….k_{34}=10

es fácil ver en la matriz de adyacencia, qué posiciones de esta se corresponden con las del vector:

	j=0	1	2	3	4
i=0	-	k_{01}=0	k_{02}=1	2	3
1	-	-	4	5	6
2	-	-	-	7	8
3	-	-	-	-	9
4	-	-	-	-	-

3.1.1.1 Paso de matriz a vector

$$k_{ij} = ni + j - \frac{(i+1)(i+2)}{2}$$

3.1.1.2 Paso de vector a matriz

El proceso inverso consiste en: dada una posición (k_{ij}) dentro del cromosoma, obtener los identificadores de los nodos (*i* y *j*) extremos de la arista correspondiente a dicha posición. Se resuelve con el siguiente procedimiento presentado en forma de pseudo-código:

```
Proc(in:n, in:k, out:i, out:j)
/*  n es el número de nodos del grafo, k la posición en el vector, i y j son los identificadores de los nodos extremos de la arista */

Var
Integer cota_menor;
Integer nro_elems_rango;
Begin
```

```
cota_menor =0;
i =0;
while(i <= n-2) loop
/*Se calcula el número de elementos del rango en que se está: primeros n-1 eltos del
array corresponden a la fila 0 de la matriz, segundos n-2 eltos: fila 1,…….último elto: fila n-
2*/

    nro_elems_rango=n-1-i;

        //Si k está dentro del rango actual en el que se está

        if (k >= cota_menor) AND (k <= cota_menor + (nro_elems_rango-1))

                j = i + (k – cota_menor +1);

                //j siempre está dentro del rango: i+1…n-1

                break loop;
      else
                cota_menor = cota_menor + nro_elems_rango;
      end if
      i = i + 1;

end loop
end proc
```

3.2 Población inicial

Para la obtención de la población inicial del algoritmo genético, se tuvo tres posibles metodologías para emplear y se detallan a continuación:

Metodología 1 (estructura a partir de un MST):

1. Un árbol de expansión mínimo (MST) se crea a partir de un nodo escogido aleatoriamente.
2. Arcos elegidos aleatoriamente (que aún no están en el árbol) son agregados para incrementar conectividad
3. Realizar chequeo de factibilidad para ese individuo (restricción de bi-conectividad y envío de flujo a través de dos caminos arista disjunto para todo requerimiento)
4. Repetir desde 1 para un nuevo nodo aleatorio hasta obtener la población inicial

Esta metodología es una modificación de la presentada por Dengiz y otros (Dengiz, Altiparmak, & Smith, 1997), pero luego de ser probada fue descartada. Al tener un único árbol mínimo de expansión para cada grafo, lo cual adolecía de generar individuos que partían de una misma topología inicial, dejando así menos variedad en la población.

Metodología 2 (estructura a partir del grafo original):

1. Tomar el grafo *G* de partida
2. Eliminar hasta un 10% de aristas del grafo original
3. Ídem a Metodología 1, parte 3
4. Repetir desde 1 hasta obtener la población inicial

La metodología presentada por Nesmachnow y otros (Nesmachnow, Cancela, & Alba, 2004) fue la escogida para el algoritmo genético. De esta forma se pudieron obtener poblaciones iniciales con una mayor diversidad de topologías factibles, fruto de la aleatoriedad con que se obtienen. Se optó por una cantidad un 5% mayor de aristas a eliminar a la usada por Nesmachnow, para permitir una más amplia variedad en problemas de tamaño más pequeño.

Metodología 3:

1. Generar un cromosoma aleatorio
2. Ídem a Metodología 1, parte 3
3. Repetir desde 1 hasta obtener la población inicial

Esta metodología fue descartada desde un principio, por considerar que la creación de un cromosoma aleatoriamente podría traer lugar a varios descartes seguidos de individuos tentativos, dada la gran cantidad de restricciones a satisfacer por el mismo (si las aristas del cromosoma generado están dentro de la solución inicial, restricciones de confiabilidad, etc.)

3.3 Función de Fitness

La función de fitness evalúa el costo del grafo representado. Se utilizará para esto la función objetivo definida en el modelo algebraico del problema (sección 1.1.1). El problema busca la minimización del costo de diseño determinístico y la esperanza de los costos operativos, considerando todos los escenarios.

3.3.1 Función

$$mín_{x,y} \sum_{(i,j)\in A} f_{ij}x_{ij} + \sum_{s\in S}\sum_{k\in K}\sum_{(i,j)\in\in A} p_s c_{ij} y_{ij}^{ks}$$

3.4 Algoritmo Genético

El algoritmo genético tiene la forma general de cualquier esquema clásico para estos. Este en particular utiliza un esquema como el presentado por Nesmachnow y otros, con la adaptación de una función para la determinación de factibilidad en los nuevos hijos obtenidos. En las secciones siguientes se detallará las decisiones tomadas tanto para los elementos básicos del AG así como los procedimientos auxiliares para reparar y asegurar la factibilidad de las soluciones.

El algoritmo genético se desarrolla de la siguiente manera:

Variables

P – Conjunto de topologías de grafo, que conforman la población

generación – entero que contabiliza el número de generaciones que se vayan dando

Constantes

g_{max} – Máximo número de generación a la que se permite llegar

Begin

1. Inicializar P(0) //Generar la población inicial, con un cierto tamaño
2. generación = 0
3. Evaluar(P(0)) //Se calcula el valor de fitness para cada individuo
4. Mientras (generación < g_{max}) Loop
    a. Padres = selección(P(generación)) /* Se seleccionan los individuos que servirán para reproducción */
    b. Hijos = Operadores_de_reproducción(Padres)
    c. Hijos_factibles = Determinar_factibilidad(Hijos) /* Se determina factibles aquellos que presenten un nivel de confiabilidad de 2 caminos arista-disjuntos para cada requerimiento y que el flujo pueda ser ruteado a través de esa topología */
    d. Nueva_pob = Reemplazar(Hijos_factibles,P(generación)) /* Se crea una nueva población con los nuevos individuos generados y otros ya existentes */
    e. generación++
    f. P(generación) = Nueva_pob
    g. Evaluar(P(generación))
5. Fin Loop
6. Retornar Mejor solución hallada

End

3.5 Selección

Para la selección de padres que se utilizaron en la reproducción se usó una selección por ruleta, donde se asigna mayor probabilidad de ser escogidos a los individuos más aptos.

3.6 Operadores de reproducción

Los operadores de reproducción usados para cruzamiento y mutación son los clásicos que se presentan dentro de un algoritmo genético, debido a la forma simple del cromosoma. Con estos operadores se obtuvieron nuevas soluciones a partir de las ya

existentes, las cuales tenían algunas de las aristas ya presentes en los progenitores y/o nuevas.

Como se recomienda en (Coello Coello, 1995) la probabilidad de cruzamiento fue del 80%, mientras que la de mutación 5%, manteniéndose así una mayor relación de realidad con la evolución biológica, la cual presenta reproducción de forma bastante frecuente, mientras que la mutación, responsable de traer diversidad al género, ocurre de manera más esporádica.

3.6.1 Cruzamiento

Se utilizó cruzamiento en dos puntos del cromosoma, elegidos aleatoriamente entre los 2 progenitores, intercambiando los genes a ambos lados de estos puntos.

3.6.2 Mutación

Para la mutación, se escogió aleatoriamente una posición (*h*) dentro del cromosoma. Si el valor en ella es 1, se elimina el arco correspondiente, haciendo valer 0 el gen. En caso de que el valor sea 0 y VAA[*h*] =1 (arista válida en el vector de aristas admitidas, sección 3.1), se agrega el arco que corresponde a tal gen, en caso que VAA[*h*] =0 (arista no válida), se vuelve a sortear una nueva posición.

3.7 Metodologías para manejo de las restricciones

Para el cumplimiento de las restricciones a las que está sujeto el problema, se desarrollaron dos algoritmos y una heurística, los cuales son sometidos a cada candidato a nuevo integrante de la población. El primer algoritmo de reparación presentado, es la primera verificación a la cual se somete el individuo, luego se ejecuta el segundo algoritmo propuesto en esta sección (el algoritmo voraz para encontrar caminos arista disjuntos). Finalmente, si el individuo cumple satisfactoriamente las restricciones de confiabilidad hasta el momento verificadas, se le somete a la heurística encargada del ruteo de flujo para cada escenario diferente, presentada por último en esta sección.

3.7.1 Algoritmo de reparación para 2 arista-conectividad

Como la 2 arista-conectividad es una propiedad de redes altamente confiables, se exige que los nodos origen y destino correspondientes a cada requerimiento presenten grado de arcos mayor o igual a 2 (sean extremos de 2 o más aristas diferentes). Por lo tanto, si para algún requerimiento con nodos *o* y *d,* origen y destino respectivamente, el grado de *o* y/o *d* es menor que 2, se aplica un algoritmo de reparación a la topología del individuo que se esté evaluando.

La estrategia de reparación es básicamente un procedimiento ávido de agregado de arcos. Para tratar de mantener el algoritmo lo más parecido al propuesto por Dengiz y otros se optó por dejar que al finalizar su ejecución, se tenga la garantía de que todo nodo

presente en la solución tenga grado mayor o igual a 2, sin tener problemas de agregado de arcos que encarezcan el costo fijo, ya que al culminar la heurística para el ruteo de flujo, se eliminan los arcos que no son utilizados. Contar con 2 arista-conectividad para cada nodo además facilita la existencia de más caminos arista disjuntos.

3.7.1.1 Notación

$\boldsymbol{N_h}$ – Conjunto de nodos de grado h
$\mathbf{N_{min}}$ – Conjunto de nodos con grado mínimo, excepto aquellos de grado 1
$\boldsymbol{n_h}$ – Cardinalidad de N_h
$\boldsymbol{m_{1j}}$ – Etiquetas de nodos en el conjunto N_1
$\boldsymbol{m_{minj}}$ – Etiquetas de nodos en el conjunto N_{min}, j = 1,2, … ,n_{min}
$\boldsymbol{f_{ij}}$ es el costo fijo de instalación para el arco (i,j)

3.7.1.2 Algoritmo

1. Determinar N_h, n_h; h = 1,…,max. grado de nodo
2. Ordenar crecientemente todos los N_h y n_h excepto N_1 y n_1, desde h = 2,..max. grado de nodo. Determinar N_{min} y n_{min}
3. Mientras $n_1 > 0$ Loop
    a. Si $n_1 = 1$, $N_1 = \{ i \}$ determinar cual conexión entre este nodo y los nodos de N_{min} tienen mínimo costo fijo ($f_{ij} \mid j \in N_{min}$) y conectarlos; $n_1 = 0$
        i. Si no existe conexión alguna válida para ningún nodo en N_{min}, repetir a con $N_{min} = N_h$, siendo N_h el conjunto siguiente en orden a N_{min}
    b. Si $n_1 >= 2$ elegir 2 nodos de n_1, denominarles m_{11} y m_{12}
        i. Si (m_{11}, m_{12}) es una arista válida y no existe ya en la solución
            1. Calcular el costo fijo f_{m_11,m_12}
            2. Calcular todos los costos f_{m_11,m_minj}, para j = 1,2,…,n_{min} tal que (m_{11}, m_{minj}) es una arista permitida y no existe ya en la solución. Denótese $m1$ = min(f_{m_11,m_minj})
                a. Si no se puede conectar m_{11} con ningún m_{minj} repetir 3.b.i.2 con $N_{min} = N_h$, siendo N_h el conjunto siguiente en orden a N_{min}
            3. Repetir lo mismo que en 3.b.i.2 para m_{12}, sea $m2$ = min($f_{m\_12,\ m\_minv}$) para v = 1,2,…n_{min}
                a. Si no se puede conectar m_{12} con ningún m_{minv} repetir 3.b.i.3 con $N_{min} = N_h$, siendo N_h el conjunto siguiente en orden a N_{min}
            4. Si $f_{m\_11,\ m\_12} < m1 + m2$ agregar la arista (m_{11}, m_{12})
            5. Sino agregar las aristas (m_{11}, m_{minj}) tal que $m1$ = min($f_{m\_11,\ m\_minj}$) y (m_{12}, m_{minv}) tal que $m2$ = min($f_{m\_12,\ m\_minv}$)
            6. $n_1 = n_1 - 2$ y volver a 3
        ii. //(m_{11}, m_{12}) no es una arista válida en la solución

1. Calcular todos los costos f_{m_11,m_minj}, para j = 1,2,…,n_{min} tal que (m_{11}, m_{minj}) es una arista permitida y no existe ya en la solución. Agregar la arista (m_{11}, m_{minj}) tal que $f_{m\_11,\ m\_minj}$ sea el de mínimo valor
    a. Si no se puede conectar m_{11} con ningún m_{minj} repetir 3.b.ii.1 con $N_{min} = N_h$, siendo N_h el conjunto siguiente en orden a N_{min}
2. Repetir lo mismo que en 3.b.ii.1 para m_{12}.

4. Fin Loop

3.7.2 Algoritmo voraz para caminos arista disjuntos

Para cumplir los requerimientos de confiabilidad, que requiere la existencia de al menos 2 caminos arista disjuntos entre todo par de nodos origen y destino para cada requerimiento, se aplicó a toda solución un algoritmo voraz que determina cuales son estos caminos. Para la obtención de cada camino individual, se utilizó el algoritmo de Dijkstra. Como valor de "largo" para todo arco, se suma su costo fijo más el variable por unidad de flujo. Una vez obtenido un camino, las aristas de este son retiradas del grafo temporalmente para poder aplicar de nuevo Dijkstra sobre la nueva topología, y así sucesivamente hasta obtener un número máximo de caminos para el requerimiento. En caso de que para algún requerimiento, no se encuentren más de dos caminos arista disjuntos, el algoritmo se detiene, considerando a la solución como no factible.

Se utilizó un parámetro con un valor para la cantidad máxima de caminos a encontrar para cada requerimiento; durante la experimentación, se determinó que el mejor valor para este, era 5, ya que un valor menor (2,3 o 4) dejaba una pequeña cantidad de caminos para usar durante el ruteo, lo cual llevaba a desechar muchas soluciones (todas inclusive en algunos problemas con gran cantidad de requerimientos), mientras que un valor más grande (7, 8 por ej.) dejaba encontrar soluciones factibles con más facilidad, pero de altos costos.

A continuación se presenta el algoritmo:

Valores de entrada:

G = (V,A) **–** Grafo sobre el que se aplica el algoritmo

K $\subseteq NxN$ es el conjunto de requerimientos entre pares de nodos origen-destino. **k**. es un miembro de *K*, con un par de nodos origen y destino asociados. Los identificadores de los requerimientos es un valor natural que comenzará en cero y se irá incrementando en uno.

H – Cantidad máxima de caminos a encontrar para cada $k \in K$

Variables:

p – Camino encontrado entre un par de nodos *v,w* dados

$\boldsymbol{P_k}$ – Conjunto de caminos arista disjuntos para cada $k \in K$

cams_hallados – Entero que cuenta la cantidad de caminos encontrados para cada requerimiento

encontrado – Booleano utilizado en la condición de parada de la segunda iteración

A' – Subconjunto de aristas de *A*

G'=(V,A') – Subgrafo de *G*, con A' como conjunto de aristas
Salida:
P = { P_k } ∀ *k* = 1,….|*K*| - Conjunto total de caminos encontrados

Valido – Booleano que vale verdadero si ∀ $P_i \in P$ $|P_i| \geq 2$ //Existen 2 o más caminos arista disjunto para los pares de nodos origen-destino de cada requerimiento.

Comenzar:

1. Para cada requerimiento $k \in K$ loop
    a. A' = A //Arcos factibles para construir un camino
    b. *G'* = (*V, A'*)
    c. Encontrado = true
    d. cams_hallados = 0
    e. Mientras (encontrado) y (cams_hallados < H) loop
        i. p = Dijkstra(*G'*, o_k, d_k) //Encuentra el camino más corto entre los nodos origen (o_k) y destino (d_k) del requerimiento *k*
        ii. si p es vacío //No existe camino entre o_k -> d_k
            1. encontrado = false //Termina la búsqueda de caminos para el requerimiento *k*
        iii. //Si existe un camino entre o_k -> d_k
            1. cams_hallados++
            2. A' = A' – {*l* | *l* ∈ p} //Se retiran los arcos del camino p
            3. Agregar p a P_k
    f. Fin loop
    g. Si cams_hallados < 2
        i. Valido = false
        ii. Terminar el algoritmo, no existen 2 o más caminos arista-disjunto para los nodos o_k y d_k
    h. //Existen 2 o más caminos arista-disjunto para los nodos o_k y d_k
        i. Agregar P_k a P
2. Fin loop
3. Valido = true //Finalizó el procedimiento, el grafo cumple la restricción de confiabilidad

Fin

3.7.2.1 Algoritmo de Dijkstra

El mismo fue implementado en su forma clásica (Dijkstra, 2008), con una pequeña modificación, que permite la detención del algoritmo en caso que el grafo pasado no sea conexo y los nodos origen y destino se encuentren en componentes conexas diferentes (condición D[s] ≠ ∞ en paso 4)

Notación:
G=(V,A) – Grafo sobre el que se aplica el algoritmo

a – Vértice de origen, $a \in V$
z – Vértice de destino, $z \in V$

C – Conjunto de nodos, que contiene los vértices de ***V*** cuyo camino más corto desde "a" todavía no se conoce.

D - Vector, con tantas dimensiones como elementos tiene V, y que guarda las distancias entre *a* y cada uno de los vértices de V.

T - Vector con las mismas dimensiones que D, y que conserva la información sobre qué vértice precede a cada uno de los vértices en el camino.

El algoritmo para determinar el camino de costo mínimo entre los vértices *a* y *z* es:

1. C = V
2. Para todo vértice $i \in C$, $i \neq a$, establecer $D[i] = \infty$; $D[a] = 0$
3. Para todo vértice $i \in C$ establecer $T[i] = a$
4. Se obtiene el vértice $s \in C$ tal que no existe otro vértice $w \in C$ tal que $D[w] < D[s]$ y $D[s] \neq \infty$
    - Si $s = z$ entonces terminar el algoritmo.
    - Si no se obtiene ningún vértice s, terminar el algoritmo devolviendo un camino vacío
5. Eliminar de C el vértice *s*: $C = C - \{s\}$
6. Para cada arista $e \in A$, que une el vértice *s* con algún otro vértice $t \in C$, de costo variable c_{st} y fijo f_{st}
    - Si $f_{st} + c_{st} + D[s] < D[t]$, entonces:
        1. Establecer $D[t] = f_{st} + c_{st} + D[s]$
        2. Establecer $T[t] = s$
7. Regresar al paso 4

Al terminar este algoritmo, en D[*z*] está guardada la distancia mínima entre *a* y *z*. Por otro lado, mediante el vector T se puede obtener el camino mínimo p: en T[*z*] está *y*, el vértice que precede a *z* en el camino mínimo; en T[*y*] está el que precede a *y*, y así sucesivamente, hasta llegar a *a*.

3.7.3 Heurística para el ruteo de flujo en la red

Se presenta a continuación una heurística que resuelve el ruteo de la demanda a enviar para cada requerimiento entre pares de nodos origen-destino (multi-commodity flow). En caso de que una topología no cumpla este requerimiento, se considera como solución no válida.

La idea de esta heurística se basó y adaptó para el problema, de la metodología presentada por Cox y otros. Si bien la codificación allí usada para el cromosoma resulta

bastante incómoda para manejar (como ya se discutió en el capítulo 2, sección 2.3), se pudo adaptar un procedimiento utilizado para la evaluación y "reparación" del cromosoma usado en éste, extrayendo las ideas principales para ruteo de demandas.

Debido a cómo fueron escogidos los caminos por donde enviar el flujo, (algoritmo Dijkstra) se realiza un ruteo eficiente por aquellos que requieren menor costo y cumpliendo con la restricción de que debe de haber un conjunto de caminos con la capacidad suficiente para transportar la demanda entre estos 2 nodos. También se mantiene la confiabilidad de 2 o más caminos arista disjuntos por donde se realiza el ruteo del flujo, garantizando la funcionalidad de la red en caso de la falla de cualquier nodo y/o arco.

Luego de realizado el ruteo del flujo para cada escenario, la heurística asegura el cumplimiento de las restricciones de:

- Conservación del flujo (la diferencia de la demanda que ingresa en cualquier nodo por un arco, con la que sale por otro de la que es también extremo es cero, excepto para los nodos origen o destino)
- La suma de demandas existentes en un arco, para un requerimiento dado, no supera el valor de la capacidad del arco.
- Para un requerimiento y un arco dados, el flujo que cruza por el segundo, no supera la demanda asociada al primero o la capacidad máxima del arco.

Una vez finalizado todo el ruteo del flujo para cada escenario, se eliminan de la topología aquellos arcos que no son usados en ningún escenario, logrando así bajar los costos fijos de instalación.

Variables

***G*=(*V*,*A*)** - Grafo que consiste de un conjunto de nodos *V* y *A* son las aristas que forman parte de la topología del grafo, y $l_m \in$ A es un arco, $m = 1,\ldots,|A|$

M – Conjunto de aristas que no llevan tráfico en ningún escenario.
$\boldsymbol{u_m}$ - Es la capacidad máxima del arco l_m.
s – Escenario actual sobre el que se aplica el algoritmo

K $\subseteq NxN$ es el conjunto de requerimientos entre pares de nodos origen-destino. **k**. es un miembro de *K*, con un par de nodos origen y destino asociados.
S conjunto discreto de escenarios de demanda, en donde para cada escenario $\boldsymbol{s} \in S$ y requerimiento $k \in K$ se tiene un nivel de requerimiento $\boldsymbol{d^{ks}}$. Los identificadores de los escenarios se representan con un número natural de cero en adelante.

$\boldsymbol{D^s}$ - Es un conjunto de demandas punto-a-punto para el escenario *s* $\{d^{ks} \mid k \in K,\ s \in S\}$. Cada $\boldsymbol{d^{ks}}$ representa la demanda a enviar entre un nodo origen y uno destino en el escenario *s*.

$\boldsymbol{x_m}$ – Capacidad usada del arco $l_m = (i,j)$, con $i,j \in V$. $x_m = \sum_{k \in K} y_{ij}^{ks}$

$\boldsymbol{a_m}$ - Es la capacidad aún sin usar para el arco l_m.

$\boldsymbol{P_k}$ - Es el conjunto de posibles caminos entre pares de origen-destino *k*, los cuales fueron determinados por el algoritmo de Dijkstra. Cada $\boldsymbol{p_j} \in P_k$ es un conjunto de arcos formando un camino continuo entre los nodos origen y destino. $\boldsymbol{P'_k}$ es un subconjunto de P_k solamente conteniendo caminos cuyos arcos tengan capacidades sin usar mayores que cero ($a_m > 0$).

r es la cantidad actual de flujo sin rutear, que debe enviarse a través de un camino durante el algoritmo.

q es el ancho de banda disponible de un camino.

usr - Representa la cantidad actual de demanda sin rutear.

$\boldsymbol{\varepsilon}$ - Valor real pequeño y constante. $\varepsilon > 0$

Algoritmo

I. $M = A$ //Inicialmente todas las aristas no llevan tráfico
II. Para cada escenario $s \in S$

1. Para cada $l_m \in A$, establecer $a_m = u_m$, $x_m = 0$
2. Para cada demanda d^{ks}:
   a. $usr = d^{ks}$ y P'_k = los caminos de P_k conteniendo sólo arcos l_m tales que $a_m > 0$
   b. Si P'_k es un conjunto vacío, no puede satisfacerse esa demanda. Establecer $usr = 0$ y terminar el algoritmo.
   c. En caso contrario, para cada p_i en P'_k, hallar el ancho de banda disponible q de p_i (el ancho de banda de un camino p_i es el mínimo valor de a_m para cada $l_m \in p_i$). Entonces r, el flujo a enviar por p_i es el mínimo entre q y usr
      i. Si $usr = d^{ks}$ //Si todo el flujo de la demanda d^{ks} podría enviarse por un solo camino
         1. r =mínimo(q, $usr(1 - \varepsilon)$) //Se permite la existencia de un flujo alternativo de nivel εd^{ks} que se enviará en los demás caminos. Saltar al punto II.2.c.iii
      ii. r = mínimo(q, usr)
      iii. Se envía el flujo r por p_i: $a_m = a_m - r$, $x_m = x_m$ + r, $M = M - \{l_m\}$ para cada arco l_m de p_i, $usr = usr - r$
      iv. Si todos los caminos fueron procesados y $usr > 0$ entonces la solución no es factible y termina el algoritmo
      v. Si $usr > 0$ y quedan caminos sin procesar repetir 2c
      vi. *Si* $usr = 0$ repetir desde 2.a para la próxima demanda

III. //Apagar las aristas que no se utilizan para ningún escenario

Para cada $l_m \in M$: $A = A - \{l_m\}$

3.8 Reemplazo de la población

Cada vez que se genera un nuevo individuo, resultante de los operadores de cruzamiento, mutación y del algoritmo de reparación para 2 arista-conectividad, que además supera las verificaciones de factibilidad, se precisa contar con una estrategia de reemplazo de individuos ya existentes entre la población y los nuevos.

De las 3 formas fundamentales de reemplazo propuestas en (Gil Londoño, 2006) se optó por la de sustituir el individuo peor adaptado por uno nuevo, garantizando así la supervivencia de los mejores adaptados y una implementación más estática al tener un tamaño de población constante. Si bien se podría haber incluido como opción que la sustitución tuviera lugar si el nuevo individuo presentara un mejor fitness que el candidato a ser desechado, se prefirió dejar mayor amplitud al espacio de soluciones factibles y permitir azarosamente, que quizás así, los nuevos integrantes aporten soluciones mejores, fruto de los operadores de cruce y/o mutación a que se les sometan.

3.9 Condición de parada

Como criterio de parada, en un principio se optó por utilizar una condición mixta de máxima cantidad de generaciones logradas, con una medida de mejora en un cierto número de iteraciones, como se recomienda en (Gil Londoño, 2006). Sin embargo, al ir experimentando se pudo observar que la segunda condición podía traer interrupciones prematuras en el algoritmo, impidiendo el descubrimiento de mejores soluciones que se alcanzaban luego de una larga cantidad de iteraciones sin mostrar mejora alguna. Por lo tanto, finalmente se optó por usar solamente la condición de máxima cantidad de generaciones.

4 Decisiones de Implementación

El algoritmo genético fue implementado en su totalidad en lenguaje C++. Si bien se cuenta con módulos realizados como clases, no se optó por mantener una metodología orientada a objetos estricta; se logró mantener un equilibrio entre módulos implementados como estructuras (o librería de procedimientos relacionados) y módulos hechos como clases C++. El criterio tomado al momento de escoger la naturaleza de un módulo fue:

- Si el módulo requiere cierta lógica para su manejo, o algoritmos de gran importancia o interacción compleja con otros módulos, se implementa como una clase.
- Si consiste en un tipo de datos con una estructura relativamente sencilla (por ejemplo una lista) y un comportamiento sin mucha complejidad, o es una librería de procedimientos y funciones con una relación común, se implementa simplemente con una pareja de archivos cabezal y de implementación, de C++.

El resto de esta sección se dividirá en 2, una con la descripción de los Tipos Abstractos de Datos (TAD de aquí en más) y las librerías más importantes, y otra con los módulos encargados de la lógica del AG

4.1 Librerías

4.1.1 MT19937

Fue la librería utilizada para la obtención de valores seudo-aleatorios. Es el único módulo no implementado para el proyecto. Consiste en una versión portable del generador de números aleatorios *Mersenne Twister* (Mersenne-Twister, 2009).

4.1.2 Random

Módulo que hará el trabajo de *proxy* entre cualquiera que requiera un valor aleatorio y la librería MT19937; se encargará también de setear una semilla aleatoria para esta última, obtenida con la función *time* de C++. Fue implementado como una clase *singleton*, permitiendo así tener una sola instancia, haciéndose más fácil el manejo de una única semilla durante toda la ejecución. También permite el cambio de semilla mientras la ejecución, algo que lograría evitar caer en ciclos de números seudo-aleatorios generados por MT19937 para ejecuciones con generaciones largas; mas cabe aclarar que no se realizaron pruebas con la puesta en práctica de esta última idea, por no considerarlo tan relevante.

4.1.3 Distancia

Esta librería calcula la distancia de aristas distintas entre la solución óptima encontrada por el algoritmo genético y la solución óptima MIP, o en su defecto la solución correspondiente límite de éste último. También calcula la distancia relativa, que es el resultado de dividir el valor mencionado anteriormente entre el total de aristas de la topología del grafo inicial.

4.1.4 Serializer

Esta librería se encarga de la lectura y escritura de datos en archivos de texto.
Se tiene un solo archivo de entrada obligatorio, con la información del grafo inicial, las demandas y los escenarios. También puede existir otro archivo (su no presencia no afecta el comportamiento del AG), únicamente con las aristas de la solución óptima encontrada por el solver MIP, lo cual sirve para las medidas de distancia de aristas en común entre la solución encontrada y la de éste último.

Al final de la ejecución se producen 2 archivos de salida, uno con la información del mejor grafo encontrado (valor de fitness, costo total fijo, total variable, aristas, etc.) y las demandas y escenarios a los que fue sometido. El otro archivo de salida contiene el valor de fitness de los mejores individuos de la población para cada generación.

Formato del archivo de entrada:

;Comentarios comienzan con ';'
Nodos = (cantidad de nodos)
Arcos = (cantidad de arcos)
Demandas = (cantidad de demandas)
Escenarios = (cantidad de escenarios)

;para cada nodo: indetificador, coordenada x y
Id x y

:
:

; para cada arista:

idNodo1 idNodo2 capacidad costoFijo costoVariable

:
:

; para cada par de demanda
id idNodoOrigen idNodoDestino

:
:

; para cada escenario

id probabilidad demandaPar1 demandaPar2 . . . demandaParN

:
:

Formato del archivo opcional de entrada:

idNodo1Arista1 idNodo2Arista1
idNodo1Arista2 idNodo2Arista2
:
:

idNodo1Aristai idNodo2Aristai

Formato del archivo de salida 1:

Val. Opt. = (Valor de fitness de la mejor solución)
Nodos = (cantidad de nodos)
Arcos = (cantidad de arcos)
Demandas = (cantidad de demandas)
Escenarios = (cantidad de escenarios)
Epsilon = (valor real para la restricción de confiabilidad)
Tiempo(s) = (tiempo que llevó la ejecución)
Costos fijos = (suma de los costos fijos para cada arista de la solución)
Costos variables = (Val. Opt. – Costos fijos)

;para cada nodo: indetificador, coordenada x y
Id x y

:
:

; para cada arista:

idNodo1 idNodo2 capacidad costoFijo costoVariable

:
:

; para cada par de demanda
id idNodoOrigen idNodoDestino

:
:

; para cada escenario

id probabilidad demandaPar1 demandaPar2 . . . demandaParN

:
:

Formato del archivo de salida 2:

Generacion 0: (Mejor valor de fitness para la generación indicada)

Generacion 1: ….

:

:

Generacion i: ….

4.2 TADs

4.2.1 Vectores

En muchos casos se tuvieron datos que eran consultados en varias ocasiones, pero que no eran modificados nunca, lo cual llevó a implementar vectores como estructuras para almacenar estos. Los que se presentan a continuación, tienen la particularidad de ser utilizados por más de un módulo:

- **Demandas**: se usó un vector de tamaño igual a la cantidad de demandas existentes. Los pares de nodos (el origen y el destino) que estaban asociados a un requerimiento, son guardados en la posición correspondiente al identificador de la demanda.
- **Escenarios**: se usó una matriz con tantas filas como escenarios existieran, y tantas columnas como demandas. Un valor real guardado en la posición [*i,j*] de la matriz corresponde al valor de la demanda a enviar para el requerimiento *j* en el escenario *i.* Se contó además con una última columna, adicional, para guardar la probabilidad de ocurrencia del escenario.

4.2.2 Vector Población

Como estructura para guardar los individuos de la población del AG, se implementó éste módulo, que tiene un *array* de tamaño igual al de la población máxima ingresada por el usuario, sin embargo, como el módulo **Inicializador** (presentado más adelante) podría llegar a encontrar menos individuos de la cantidad deseada, se utiliza un campo que guarda el verdadero tamaño de la población hallada, que puede ser menor que el anterior mencionado, controlando así que no se acceda a casillas del vector que tengan valores nulos. Cada posición del *array* tendrá un Cromosoma y un campo real con el valor de fitness correspondiente al individuo, como se muestra en la Figura 7.

El modulo cuenta con un procedimiento para ordenar los individuos por su valor de fitness, dejando al peor adaptado en la primera posición del vector, y a la mejor solución en la última. Está realizado por ordenación de burbuja (Bubble-Sort, 2009), revisando cada individuo del *array* con el siguiente, e intercambiándolos de lugar si el primero es mejor que el segundo.

Con la estructura de un vector, con elementos ordenados dentro suyo, se permite acceder a los individuos peor adaptados para reemplazarse con $O(1)$ peor caso, así como también la mejor solución encontrada; por otro lado, la selección por ruleta devuelve un índice del *array* en el **Vector Población** con el individuo escogido (siendo éste un acceso que se da muy seguido durante cada iteración el AG). Las inserciones realizadas durante la inicialización también se realizan con ese mismo costo, ya que alcanza con ir incrementando en uno el valor del campo que guarda el tamaño de población encontrada, para tener la posición nueva donde guardar el individuo. Se tiene la cadencia de la

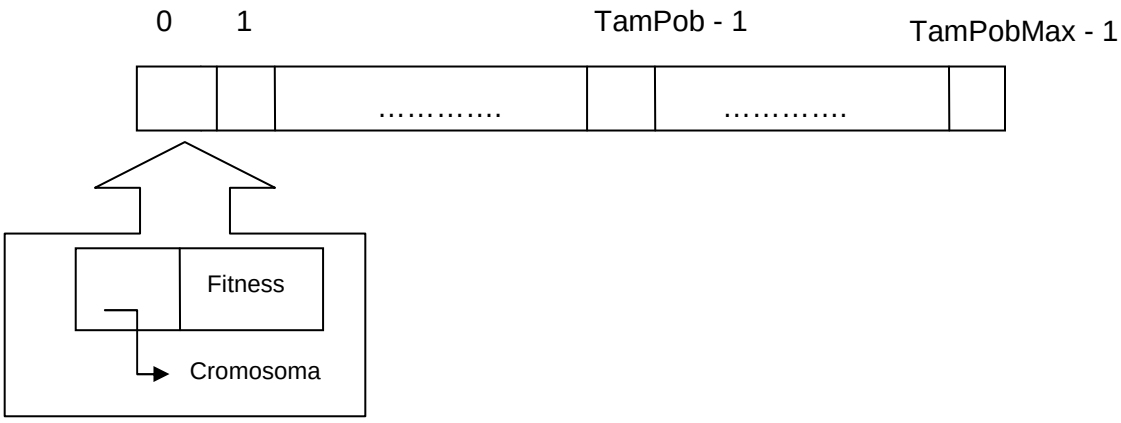


Figura 7: Vector Población

ordenación de burbuja, la cual tiene *O*(TamPob^2) al realizarse, aunque esto ocurre una vez por cada generación.

4.2.3 Vector de Caminos

Es un TAD con una estructura formada a partir de otros TADs. Su estructura consiste en un *array* de tamaño igual al del vector de requerimientos, conteniendo cada casilla un conjunto de caminos arista disjuntos entre los nodos origen y destino asociados al requerimiento, como se puede ver en la Figura 8. El identificador de la demanda se corresponderá al número de la posición en el vector.

Se optó por esta estructura ya que las inserciones, y la recuperación de los caminos para cada requerimiento son las únicas operaciones que se realizan sobre éste TAD, permitiéndose así que estas funcionalidades se realicen con *O*(1) para el peor caso. Por otra parte, la estructura escogida para los conjuntos de caminos arista disjuntos fue el de una cola FIFO (*First in, First out*). Cada elemento de la cola, tiene una lista de nodos (implementada como una lista de enteros) que conforman un camino, también cuenta con un campo de valor real, cuya funcionalidad se explicará en el módulo **Flowrouter**. Este TAD FIFO es necesario debido a que el algoritmo voraz de **Gafodilp** (módulo presentado más adelante, implementa el algoritmo ávido para hallar caminos) va encontrando los caminos aristas disjuntos en orden creciente de costo, y el flujo va siendo enviado por aquellos que tengan menor coste, lo cual lleva a que el primer camino encontrado sea más tarde el primero en recuperarse para el ruteo, luego el que le sigue y así sucesivamente.

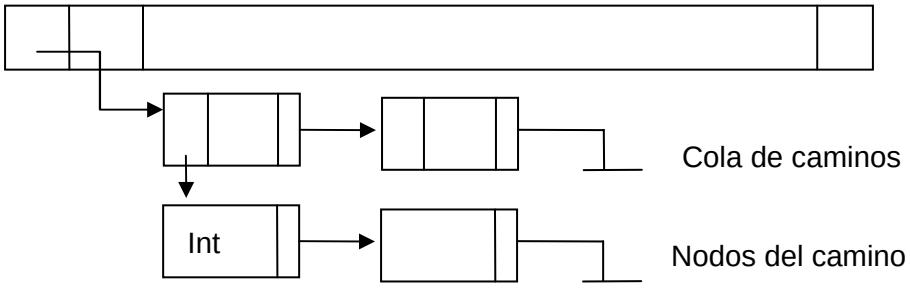


Figura 8: El vector de caminos

4.2.4 Cromosoma

Este módulo posee la estructura con la que se representa el cromosoma, un atributo entero con el número de nodos del grafo, además de un grupo de métodos de utilidad para trabajar con el mismo.

La estructura que mejor se asemeja a la forma del cromosoma escogida para el AG es un vector de valores booleanos (1 si el arco correspondiente a ese casillero está presente en la solución, 0 si no) de tamaño $n(n-1)/2$, siendo n el número de nodos existentes en el grafo. Tanto en la mayor parte del AG como en los algoritmos complementarios, se manejan los arcos por su posición dentro del cromosoma, más que por los identificadores de sus nodos extremos, por lo que se tiene un mejor tiempo de acceso a los genes con esta estructura que con cualquier otra, además de permitir mantener las operaciones de cruzamiento y selección tal como se definen en la teoría.

4.2.5 Grafo

El grafo fue implementado como una **lista de adyacencia**, siendo esta opción más practica que la de una **matriz de adyacencia**. Esta decisión se basó en la topología de las redes con las que se iba a estar trabajando, redes con cantidades de arcos que no llegarían a la mitad de los existentes si fuera completo.

Una matriz de adyacencia reservaría lugar para muchos casilleros que rara vez se utilizarían, además de que una función para obtener todos los arcos adyacentes a un nodo (que ocurre de forma bastante repetida, como por ejemplo con Dijkstra) llevaría a tener que recorrer toda una fila de la matriz, mientras que con listas de adyacencia, esa función se realizará con la recorrida de la lista apropiada.

Como las operaciones de agregado y eliminación de nodos casi que no se llevan a cabo (solamente al ir creándose el grafo la primera vez), y las de agregado y eliminación de aristas se realizan sobre el cromosoma, muchas veces sin necesidad de reflejarlo en la representación del grafo, estas operaciones de mayor carga computacional en la representación escogida, tienen poca ocurrencia. En la Figura 9 puede verse un ejemplo de uso de listas de adyacencia para un grafo no dirigido.

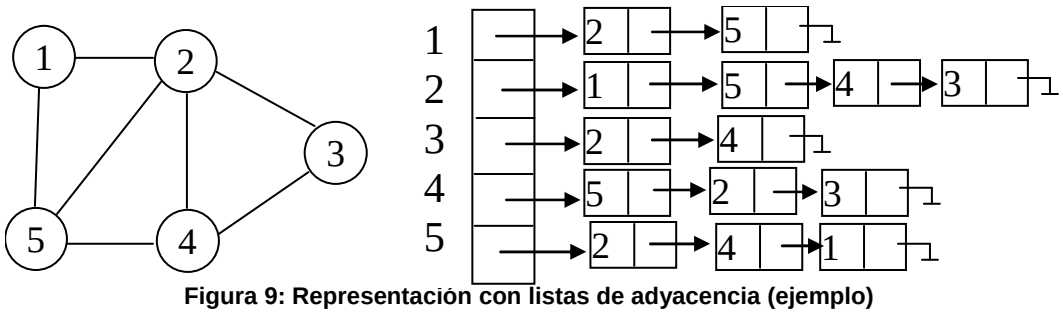


Figura 9: Representación con listas de adyacencia (ejemplo)

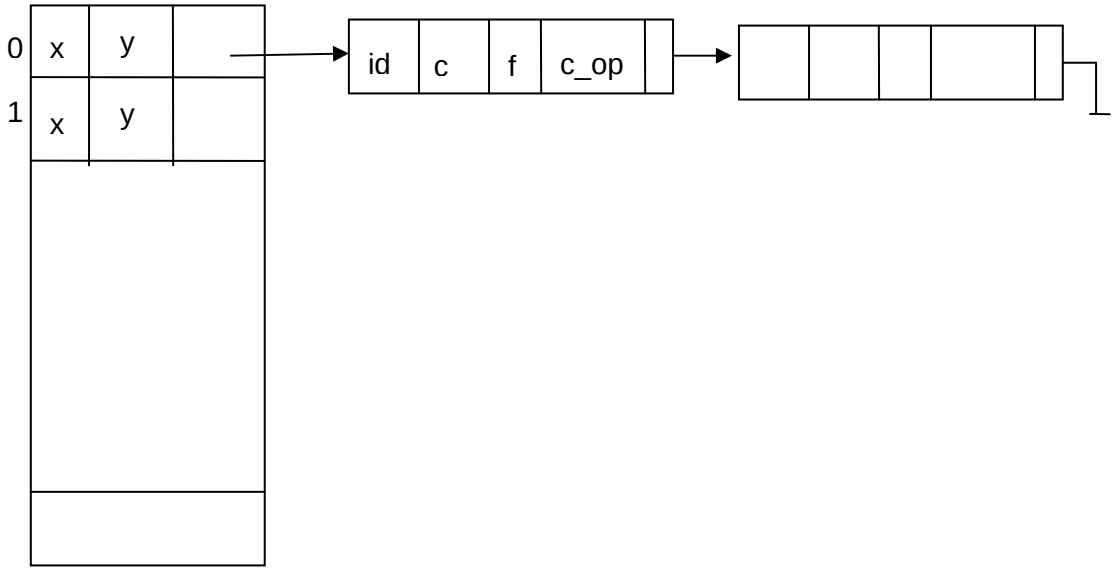


Figura 10: Estructura del Grafo

Como los identificadores de los nodos son valores naturales crecientes, comenzando desde cero, se usó un arreglo de celdas de cabecera, de tamaño igual al número de nodos del grafo, correspondiéndole cada número de casillero al nodo de igual identificador. Como se muestra en la estructura de la Figura 10, cada celda contiene dos atributos correspondientes a las coordenadas *X* e *Y*, ambas de valor real, además de una lista con los nodos adyacentes. Cada elemento de esta lista contiene cuatro atributos: el identificador del nodo adyacente, la capacidad, el costo fijo de instalación y el operativo del arco que conforman ambos nodos.

4.3 Módulos

Los módulos que se presentan a continuación fueron los encargados de toda la lógica más pesada de la implementación; como ya se mencionó anteriormente, estos se implementaron como clases.

Los módulos **R2Conectividad**, **Gafodilp**, **Flowrouter** y **Factibilidad**, son los utilizados para la evaluación de factibilidad de cada posible solución. Los módulos **Inicializador** y **Metaheuristica** estuvieron encargados de la lógica del algoritmo genético y finalmente, **Problema** fue implementado como módulo principal, a cargo de las inicializaciones necesarias, la invocación de la ejecución del AG, la interacción con el usuario, etc.

4.3.1 R2Conectividad

Contiene el algoritmo que se encarga de mantener la restricción de 2 arista-conectividad en el grafo, para aquellos nodos que tengan grado 1 de aristas.
Para su funcionamiento este módulo cuenta con:

- **El Grafo original**, o sea, el grafo inicial del problema para el que se está buscando un óptimo.
- **El VAA** (Vector de aristas admitidas, se implementó como el Cromosoma representativo de la topología del grafo original)
- **El Cromosoma de la solución siendo evaluada**
- **Un Vector de Grados**: contiene conjuntos de nodos, agrupados por el grado de conectividad que tengan estos.

El **Vector de Grados** mencionado, es un TAD implementado con la estructura de un vector de listas de enteros, de tamaño $k = n(n\text{-}1)/2$, con n igual al número de nodos del grafo, ya que este valor k es el tamaño de arcos en un grafo completo y un nodo no podrá tener más de k-1 aristas.

Si un nodo tiene un grado h de vértices adyacentes, entonces, este se encuentra en la lista situada en la posición h del vector. Esto permite una facilidad para el algoritmo de reparación de 2 arista-conectividad, ya que se pueden obtener todos los que tengan mismo grado con solo recuperar la lista de la posición correspondiente, tarea muy recurrente durante el algoritmo, pero con un peso operacional al momento de actualizarla (agregar una nueva arista, cambiar los nodos de conjunto).

4.3.2 Gafodilp

El módulo Gafodilp (Greedy Algorithm for Disjoin Link Paths) tiene la implementación del algoritmo voraz para encontrar caminos aristas disjuntos para cada demanda. Los caminos se encuentran aplicando el algoritmo de Dijkstra.
Para su funcionamiento, este módulo cuenta con:

- **Un grafo**, el cual tiene la topología de la posible solución factible que se está evaluando, para la cual se quieren encontrar los caminos.
- **El vector de requerimientos**
- **Un Vector de Caminos**

4.3.3 FlowRouter

Contiene una heurística que se encarga de realizar el ruteo de flujo para cada escenario, a través de la red. Aquellos arcos que no son usados en ningún escenario, son eliminados del cromosoma que se devuelve al final. También calcula el costo del flujo que se envía para todos los escenarios, por lo que realiza la siguiente parte del cálculo del valor de fitness: $\sum_{s \in S} \sum_{k \in K} \sum_{(i,j) \in \in A} p_s c_{ij} y_{ij}^{ks}$; esta función representa la esperanza del costo operativo, segmento parcial de la función objetivo definida en el modelo algebraico del problema (sección 1.1.1). El cálculo realiza la suma del costo operativo (c_{ij}) por unidad de flujo en cada arista (y_{ij}^{ks}) para todo escenario $s \in S$, por la probabilidad de ocurrencia de éste (p_s).

Para su funcionamiento este módulo cuenta con:

- **Un grafo**, el cual tiene la topología de la posible solución factible que se está evaluando.
- **El Cromosoma de la solución siendo evaluada**
- **Un Cromosoma que indicará las aristas del Cromosoma que no se utilizan**(*****)
- **El Vector de Requerimientos**
- **El Vector de Escenarios**
- **El Vector de Caminos** encontrado por el módulo *Gafodilp*.
- **Un Vector para la capacidad de los arcos**: de tamaño igual al del Cromosoma, asociando cada posición del *array* con la arista correspondiente en el Cromosoma.

En (*) se utilizó otro Cromosoma, inicializado igual que el de la solución, para determinar cuáles aristas no son usadas durante el ruteo, para ningún escenario. Los genes de las aristas que se usan, pasan a tener valor cero, para así, una vez finalizado todo el procedimiento, devolver un nuevo cromosoma, resultado de realizar un XOR gen a gen entre el de la solución y éste otro.

Como se adelantó en la sección 4.3.2, los elementos de las colas de caminos para cada demanda, tienen un campo de valor real. Este sirve para guardar allí, durante el ruteo del flujo en cada escenario, el valor del mínimo ancho de banda disponible en los arcos del camino; se entiende por mínimo ancho de banda disponible en un camino como el mínimo valor de a_m para toda arista m perteneciente a éste (sección 3.7.3).

El **Vector para capacidad de arcos** guarda dos valores reales: la capacidad usada para el arco m (x_m) y la disponible aún (a_m). Se optó por esta estructura ya que permite mantener un paralelismo entre las posiciones en el cromosoma y este vector. Los arcos se identifican en la heurística con los identificadores de sus nodos extremos, y con éstos se puede obtener la posición dentro del vector con el mismo método de conversión para el cromosoma visto en el capítulo anterior, accediendo así a la información de capacidad usada y disponible de la arista en $O(1)$ para el peor caso, la cual es una de las funcionalidades más reiterativas.

4.3.4 Factibilidad

Se encarga de determinar la factibilidad de la topología que se le asigne. Realiza la verificación de 2 arista-conectividad para aquellos nodos que forman parte del origen y destino de cada requerimiento e invoca al algoritmo de reparación para 2 arista-conectividad (**R2Conectividad**) en caso de ser necesario. Utiliza a **Gafodilp** para determinar los caminos arista disjuntos y pasa los resultados obtenidos a **FlowRouter**, determinando al final el valor total de fitness para la topología, en caso de que esta cumpla todas las restricciones del problema a las que es sometida dentro de éste módulo.
Para su funcionamiento este módulo cuenta con:

- **El Vector de Requerimientos**
- **El Vector de Escenarios**
- **El Grafo del problema original**
- **El Cromosoma de la solución siendo evaluada**

4.3.5 Inicializador

Genera una población inicial conformada de topologías distintas. Los individuos se generan eliminando el 10% de aristas de la topología del grafo original del problema. Para cada candidato a formar parte de la población, se determina si es factible o no (**Factibilidad**). Se usó un umbral para el tamaño de la población, abortando la operación en caso de que la población generada no supere la cantidad mínima, y no dejando que existan más individuos que los indicados en la cantidad máxima. La cota mínima de éste umbral es de 3 individuos, ya que una población de tamaño menor denotaría en un ciclo infinito en la ejecución del AG Esto se debe a la selección por ruleta implementada, como se verá en el módulo siguiente. La cota máxima es un valor ingresado por el usuario.

También se tuvo un parámetro que acotó la cantidad de iteraciones que se permiten hacer para encontrar topologías posibles, por ejemplo 2000 iteraciones. En la mayoría de los problemas ejecutados, este tamaño (2000) fue suficiente para encontrar la población inicial, pues se pudo notar en los problemas más pesados que eran necesarias más de 1000 iteraciones en algunas de sus ejecuciones. La idea de este parámetro surgió ante la aleatoriedad con que se buscan las soluciones iniciales, lo cual lleva a realizar varios intentos para encontrar un individuo válido; esto trae dificultades para cualquier tarea de calcular teóricamente cotas máximas para estas iteraciones.
Para su funcionamiento éste módulo cuenta con:

- **El Vector de Requerimientos**
- **El Vector de Escenarios**
- **El Grafo del problema original**
- **El VAA**
- Una instancia de la clase **Factibilidad**, a la cual se le asigna cada posible solución, a medida que se obtienen, para evaluar si cumple los requerimientos del problema o no.

4.3.6 Metaheurística

Implementa el algoritmo genético. Esta sección se enfocará más en las decisiones de uso de estructuras así como una descripción de la implementación de la selección por ruleta.
Para su funcionamiento este módulo cuenta con:

- **El Grafo del problema original**
- **El VAA**
- **El Vector de Requerimientos**
- **El Vector de Escenarios**
- Una instancia de la clase **Factibilidad**, a la cual se le asigna cada posible solución, a medida que se obtienen, para evaluar si cumple los requerimientos del problema o no.
- Un vector (de tamaño igual al máximo de generaciones), que guarda el mejor valor de fitness para cada generación. Esta información se utilizó como una forma de seguir el proceso de evolución de la población del AG para los problemas testeados.
- El **Vector Población**, con todas las soluciones
- La **Ruleta**, implementada también como un vector de tamaño igual al tamaño de la población.

Selección por Ruleta

La **ruleta** para la selección proporcional se implementó como un vector, de tamaño igual al *array* del **Vector Población**. Esta guarda en cada casillero el valor de la suma parcial de los valores de fitness para cada individuo hasta esa posición, yendo sumando del mínimo al máximo valor.

En realidad aquí no se está realizando una selección proporcional con mayor probabilidad de ser elegidos aquellos que conforman un mayor porcentaje del fitness total, sino que se modificó la idea para adaptarla al problema, teniendo una ruleta, con posiciones que se correlacionan con las del *array* en **Vector Población**, dándole más probabilidad de escogerse en la primera las posiciones que se corresponden con la de los mejores individuos en el segundo.

Ejemplo: Suponiendo tener los valores de fitness 2,2,7,3,5 para 5 individuos; como ya se explicó para el TAD **Vector Población** las soluciones estarán ordenadas en orden decreciente: 7,5,3,2,2 sumando estos un total de 19.

Las sumas parciales yendo del mínimo al máximo valor serán: 2, 2+2=4, 4+3=7, 7+5=12, 12+7=19, valores que en ese orden estarán situados en las posiciones 0, 1,..5 respectivamente del vector de la **ruleta**. En la Tabla 1 pueden verse los porcentajes que representan cada valor del total.

	Fitness	Porcentaje (%)
	2	10,5
	2	10,5
	7	37
	3	16
	5	26
Total	19	100

Tabla 1

Las diferencias entre el valor de una posición y el de la anterior se irán incrementando, dejando para las más altas una mayor probabilidad de ser seleccionadas que las bajas (19-12=7=37% del total). Se elige aleatoriamente una posición en la ruleta y se utiliza esa misma posición para obtener el individuo en el *array* del **Vector Población**, recordando que en las posiciones más altas de éste último, se encuentran los individuos de menor valor de fitness (las mejores soluciones).

A continuación se muestra el seudo-código implementado al momento del sorteo:

Variables:

randomFitness – valor real generado aleatoriamente dentro del intervalo acotado por el mínimo valor de fitness existente entre la población y la suma del total.

id – Tiene el índice de la posición elegida.

primero, medio, ultimo – Enteros usados para ir acotando el umbral de valores que son tenidos en cuenta en la ruleta.

cantIndividuos – Entero con la cantidad de individuos existentes en la población

Begin

1. randomFitness = random(ruleta[0], ruleta[cantIndividuos-1]) /*Se genera un valor aleatorio entre el mínimo fitness y la suma total de todos ellos*/
2. id = -1
3. primero = 0
4. ultimo = cantIndividuos – 1
5. medio = (ultimo – primero) / 2
6. while (id = -1 y primero <= ultimo) Loop /*Se busca quedarse con el valor más cercano al valor aleatorio generado*/
    - if (randomFitness < ruleta[medio])
        1. ultimo = medio
    - else if (randomFitness > ruleta[medio])
        1. primero = medio
    - medio = (primero + ultimo)/2
    - if ((ultimo - primero) = 1)
        1. id = ultimo
7. End Loop
8. //Devolver la posición id para el vector de población

End

4.3.7 Problema

Se le cargan todos los datos del problema (Grafo, requerimientos, escenarios-demanda) e invoca el algoritmo genético por medio de **Metaheuristica**. Despliega los resultados obtenidos por medio de la librería **Serializer**; también enseña por pantalla los resultados y mensajes de error que correspondan mostrar al usuario.

4.4 Diagrama de Módulos

En la Figura 11 se muestra un diagrama con los módulos principales del AG y cómo están relacionados entre sí. Aunque no se trata de un modelo conceptual, ni un diagrama de clases, se intenta dejar una visión más clara de cómo interactúan los módulos presentados anteriormente.

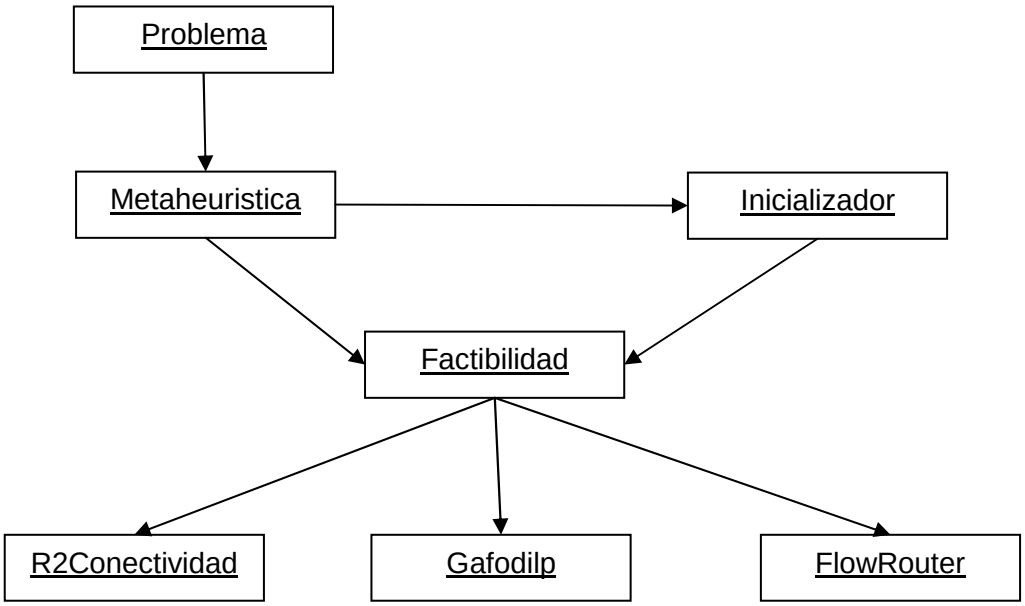


Figura 11: Diagrama de Módulos

5 Experimentos computacionales

5.1 Problemas de prueba

Se probaron 25 problemas diferentes obtenidos de Olivera y otros. Cada uno de estos problemas cuenta con:

- Una cantidad dada de nodos, con su identificador (un entero natural) y par de coordenadas (*x*,*y*) correspondientes (valores reales).
- Una cantidad dada de aristas, con los identificadores de sus nodos extremos, la capacidad (entero), el costo fijo y variable (ambos valores reales).
- Una cantidad dada de pares de demanda, con su identificador (entero natural) y los identificadores de nodos origen y destino.
- Una cantidad dada de escenarios, con su identificador (entero natural), la probabilidad de ocurrencia (valor real, entre cero y uno) y un entero natural para cada requerimiento, que será la cantidad de demanda a cumplir en ese escenario.

№	Nodos	Arcos	Demandas	Escenarios
1	20	72	4	5
2	20	72	4	5
3	20	72	4	5
4	20	72	4	5
5	20	73	4	5
6	20	73	4	5
7	20	69	8	5
8	20	69	8	5
9	20	69	8	5
10	20	81	8	5
11	20	81	10	5
12	20	81	10	5
13	30	178	12	5
14	30	178	12	5
15	30	175	12	5
16	30	175	12	5
17	30	184	12	5
18	30	184	12	5
19	30	177	12	10
20	30	177	12	10
21	30	177	12	10
22	30	192	12	10
23	30	192	12	10
24	30	192	12	10
25	40	318	12	10

Tabla 2: Listado de los 25 problemas

La Tabla 2 muestra las dimensiones de los problemas. Los primeros 5 pueden ser considerados como problemas livianos, de pocos nodos, arcos, demandas y escenarios, siendo el 5º el más “pesado” de este grupo, con escenarios teniendo valores de demanda

un poco más grandes (339 para el último requerimiento del último escenario, así como en los demás problemas, apenas se pasa de 200). Por otra parte, la capacidad de los arcos no supera 250, lo cual lleva a que los flujos deban ser llevados por más de 2 caminos en más de una ocasión.

Los siguientes 15 problemas a estos van presentando un mayor nivel de complejidad sucesivamente. La tendencia al principio es el aumento de la cantidad de demandas, para luego irse incrementando los escenarios, manteniendo el número de arcos y nodos relativamente igual. Se tiene el caso especial del número 7, un problema extremadamente pesado con 8 demandas (hasta el anterior no pasan de 4), lo que lleva a tener un gran tráfico de flujo por una red con arcos de corta capacidad (no más de 250). Luego, va aumentando el número de nodos así como el de aristas, llegando hasta 177 arcos en el problema 20. Las demandas y escenarios también van siendo más, llegando hasta 12 y 10 respectivamente.

Los últimos 5 problemas son los de mayor esfuerzo computacional para el AG con excepción del número 21, el cual presenta los mismos nodos, arcos y demandas del 20, con la diferencia de contar con más capacidad para los arcos (509 cap. máxima en el 21 contra 250 del 20) y valores distintos de demanda. Para los otros restantes, se da un aumento en el número de nodos y aristas, llegando a 40 y 318 respectivamente en el 25.

Según Olivera y otros, estos problemas fueron obtenidos por un generador aleatorio de redes, al cual se le indica el número de nodos, demandas y escenarios a tener, la capacidad mínima y máxima que pueden poseer los arcos y los parámetros para las distribuciones que darán los valores aleatorios.

Utilizando un solver *branch and bound* que implementa la formulación Mixed Integer Programming (MIP), se encontraron las soluciones óptimas para los primeros 10 problemas, mientras que para los restantes, por ser más “pesados”, se tiene el valor hallado luego de una hora de ejecución, tiempo tras el cual se interrumpió la misma. Además como datos adicionales, se cuenta con la topología de la solución, el tiempo que llevó de ejecución, el parámetro épsilon usado para el requerimiento de confiabilidad y una muestra del costo fijo y variable por separado.

Se logró comparar estos resultados con los que se obtuvieron por medio del AG, no sólo en costos, sino también en tiempo de ejecución.

Se calcularon los valores relativos de la diferencia entre los mejores costos para ambos métodos, de forma de saber cuánto superior es el resultado del AG, en porcentaje, con respecto al óptimo hallado por MIP

También se compararon los resultados, para los mismos problemas, con los obtenidos por medio del algoritmo GRASP desarrollado por Olivera y otros.

Otra medida que fue tenida en cuenta, fue la de cantidad de aristas diferentes entre la solución óptima (o más cercana) del MIP y la del AG Como ya se mencionó, el módulo **Distancia** estuvo encargado de obtener estos valores. De esta forma se tuvo una estimación de qué tan cercanas son en su topología las soluciones, partiendo de las aristas del grafo inicial del problema.

Sea X^{MIP} la solución óptima del problema MIP
Sea X^{AG} la solución óptima del algoritmo genético

X^{MIP}, $X^{AG} \in B^{|A|}$ siendo A el conjunto de aristas del grafo original y B un vector de valores booleanos.

Definimos la distancia en aristas de X^{MIP} a X^{AG} de la siguiente forma:

$$d(X^{MIP}, X^{AG}) = \sum_{(i,j)\in A} \left| X_{ij}^{MIP} - X_{ij}^{AG} \right|$$

Y la distancia relativa:

$$d\ Rel(X^{MIP}, X^{AG}) = \frac{d(X^{MIP}, X^{AG})}{|A|}$$

5.2 Parámetros para el Algoritmo Genético

Epsilon

El valor de épsilon (definido en la sección: 1.1 Definición del problema) usado para los primeros 5 problemas fue de 0,0001, mientras que para el resto, fue de 0,001.

Semilla

Fue un valor obtenido aleatoriamente, como ya se explicó en el capítulo anterior (sección 4.1.2 Random). Se muestra en pantalla la semilla generada, permitiéndose así volver a representar ejecuciones que se consideren interesantes.

Número de individuos

El número de individuos estuvo preferentemente entre 16 y 21. En (Gil Londoño, 2006) se menciona que ese tamaño fue demostrado de ser suficiente para el enfrentamiento con éxito de varios tipos de problemas, por lo que se intentó ver si aquí sirven también como útiles cantidades de población. También, se probó con poblaciones de 50 y 100 individuos, como pruebas límite, apostando a una mayor variedad dentro de la solución y quizás brindando una convergencia a valores óptimos de manera más rápida.

Cantidad de Generaciones

La cantidad de generaciones estuvo correlacionada la mayor parte de las pruebas, con el tamaño de la población, usando altos valores de la primera (15000-25000 iteraciones) para cantidades entre 16 y 21 de la segunda, mientras que para los casos extremos de 100 individuos, no se superaron las 6000 generaciones, ya que valores más altos en estos casos podían llevar a ejecuciones muy largas, las cuales era probable que estuvieran estancadas en un óptimo local desde muchas generaciones atrás; por otra parte fue una forma de permitirse estudiar el comportamiento de la variedad en la población por parte de la mutación, dando la oportunidad con un mayor número de generaciones, de presentarse una “alteración” que introdujese una mejor solución.

5.3 Resultados del Algoritmo Genético

Las pruebas se corrieron con un computador con procesador Intel Core 2 duo, 1.83GHz, y 3GB de memoria RAM.

En la Tabla 3 puede verse cada problema sometido a ejecución, su cantidad de nodos, aristas, demandas y escenarios. Para cada uno se muestra el resultado y tiempo obtenidos por el solver MIP y el algoritmo GRASP. Las últimas 4 columnas, contienen el tamaño de población, el número de generaciones, el valor objetivo y el tiempo de ejecución correspondientes al algoritmo genético.

Como se puede ver, sólo en siete problemas se encontró un valor óptimo para el AG con 100 individuos y 2500 o 6000 generaciones. Se tiene la excepción de los problemas 23 y 24 que con 600 iteraciones se tuvo el mejor valor, lo cual puede deberse a un óptimo local del cual no se pudo salir (aunque reiteradas pruebas con diferentes poblaciones y generaciones sobre estos no dieron mejores resultados), o al uso de una cantidad de individuos grande, portando una enorme variedad de soluciones, lo que llevó a una convergencia prematura. Tres problemas (20, 22 y 25) solamente obtuvieron el óptimo con 50 individuos en la población. Para el resto, se comprobó que entre 16 y 21 individuos y un número alto de generaciones (superior a 20000) es suficiente para tener buenos resultados, confirmando en buena parte lo expuesto por Alander en (Gil Londoño, 2006). Aunque para muchos problemas las gráficas de evolución muestren una convergencia en muchas menos generaciones que la máxima (Figura 14), en otras puede verse que luego de un período largo sin mejora alguna, se producen uno o dos cambios para luego volverse a estancar (Figura 15), lo cual llevó a seguir usando esa cantidad de iteraciones. Por otra parte, en pruebas realizadas con 30000, 60000, incluso hasta 80000 generaciones no se registraron mejoras por encima de las 25000, lo que llevó a dejar ésta última como cota superior.

Para el problema 7, considerado como problema “pesado”, no se encontró solución alguna factible. Para éste se probó ejecutar el AG con diferentes configuraciones (combinadas o aparte) de porcentaje de aristas eliminadas así como la cantidad de iteraciones máximas para encontrar la población inicial. Como éste problema en particular presenta escenarios con altos valores de demandas y arcos de capacidad limitada, se intentó también aumentar la cantidad de caminos arista disjuntos para cada requerimiento, lo cual podría dar lugar a más rutas con capacidad disponible para el ruteo del flujo, con la penalidad de un mayor costo variable y total; sin embargo no se tuvo ningún resultado positivo. Por otra parte, puede observarse para el problema 19 que el algoritmo GRASP no encontró ninguna solución factible, mientras que el AG sí lo logró, contrario a lo que ocurrió para el 7.

Problema					MIP		GRASP		AG			
ID	N	A	D	E	Objetivo	Tiempo(s)	Objetivo	Tiempo(s)	Población	# Generación	Objetivo	Tiempo(s)
1	20	72	4	5	18.835	5	18.831	600	18	25.000	20.920	54
2	20	72	4	5	18.686	17	18.683	600	16	10.000	20.482	17
3	20	72	4	5	21.225	32	21.337	433	17	25.000	21.225	59
4	20	72	4	5	20.787	18	20.821	372	18	20.000	21.812	51
5	20	73	4	5	29.445	82	29.723	600	17	11.000	34.014	30
6	20	73	4	5	21.773	19	21.840	584	18	15.000	23.290	33
7	20	69	8	5	40.906	3.771	44.263	450	Sin solución factible			
8	20	69	8	5	30.521	691	31.540	600	17	25.000	32.616	136
9	20	69	8	5	29.867	871	30.986	600	17	13.000	33.667	70
10	20	81	8	5	33.080	2.783	35.452	601	21	20.000	36.157	82
11	20	81	10	5	40.783	3.600	43.929	601	21	11.000	44.209	72
12	20	81	10	5	34.809	3.600	36.065	601	100	2.500	35.874	120
13	30	178	12	5	43.625	3.600	43.291	602	16	25.000	44.363	153
14	30	178	12	5	43.340	3.600	44.091	602	100	2.500	42.809	147
15	30	175	12	5	40.440	3.600	42.344	602	100	6.000	39.262	407
16	30	175	12	5	40.099	3.600	42.890	602	100	6.000	39.943	476
17	30	184	12	5	52.730	3.600	57.833	602	21	20.000	58.521	193
18	30	184	12	5	57.432	3.600	59.486	601	100	6.000	59.028	513
19	30	177	12	5	70.194	3.600	Sin solución factible		100	6.000	58.833	493
20	30	177	12	5	54.015	3.600	53.564	607	50	20.000	52.660	589
21	30	177	12	5	65.384	3.600	44.674	605	100	6.000	44.988	375
22	30	192	12	5	74.233	3.600	53.720	603	50	25.000	50.480	802
23	30	192	12	5	72.870	3.600	57.094	603	100	600	54.152	46
24	30	192	12	10	67.991	3.600	45.963	607	100	600	41.976	43
25	40	318	12	10	79.710	3.600	52.416	603	50	25.000	48.492	941

Tabla 3: Resultados[2]

Otro detalle para observar son los tiempos empleados en las tres metodologías. Para los problemas 1, 3, 4 y 6, el solver MIP encuentra la solución óptima global en mejor tiempo que GRASP y el AG. Del 7 en adelante, se puede notar una diferencia muy amplia entre los tiempos de ejecución del AG con respecto a MIP (para el problema 11 por ejemplo, vemos un minuto y doce segundos del primero contra una hora del segundo).

Con respecto a GRASP, los tiempos de éste rondan los 600 segundos en su mayoría, mientras que AG supera los 10 minutos únicamente en dos ocasiones.

También sirven como una muestra de mayor esfuerzo computacional para el AG los largos tiempos de ejecución para los últimos problemas testeados (los de mayor cantidad de arcos y requerimientos).

[2] En las soluciones de MIP, para valores de tiempo igual a 3600 se muestra el valor límite encontrado hasta ese momento

Problema	Objetivo			Distancia(MIP,AG)	
ID	GRASP/MIP(%)	AG/MIP(%)	AG/GRASP (%)	Absoluta	Relativa(%)
1	-0,02	11,07	11,09	7	10
2	-0,02	9,61	9,63	8	11
3	0,53	0,00	-0,52	0	0
4	0,16	4,93	4,76	16	22
5	0,94	15,52	14,44	10	14
6	0,31	6,97	6,64	3	4
7	8,21				
8	3,34	6,86	3,41	5	7
9	3,75	12,72	8,65	16	23
10	7,17	9,30	1,99	20	25
11	7,71	8,40	0,64	19	23
12	3,61	3,06	-0,53	16	20
13	-0,77	1,69	2,48	19	11
14	1,73	-1,23	-2,91	26	15
15	4,71	-2,91	-7,28	16	9
16	6,96	-0,39	-6,87	19	11
17	9,68	10,98	1,19	35	19
18	3,58	2,78	-0,77	30	16
19		-16,19		32	18
20	-0,83	-2,51	-1,69	34	19
21	-31,67	-31,19	0,70	40	23
22	-27,63	-32,00	-6,03	43	22
23	-21,65	-25,69	-5,15	47	24
24	-32,40	-38,26	-8,67	34	18
25	-34,24	-39,16	-7,49	44	14

Tabla 4: Resultados comparativos

En la Tabla 4 se pueden ver las comparaciones (en porcentajes) entre las tres metodologías. Hasta el problema 11 inclusive, GRASP predomina sobre AG encontrando mejores soluciones, con excepción del problema 3, para el cual éste último parece haber encontrado la solución óptima global. Del 12 en adelante, el algoritmo genético desarrollado predomina sobre los otros dos, sobre todo en los últimos cuatro problemas (los de mayor cantidad de aristas y demandas). Cabe observar, que los peores resultados se dieron en los problemas 5 y 7 (15,5% de diferencia para el primero, sin soluciones para el segundo), los cuales se encuentran entre los más “pesados”, entendiéndose por “pesados” a aquellos del conjunto de problemas con menor holgura en la capacidad de los arcos, sumando a eso escenarios con altas demandas. Por otra parte, la mejor solución se dio para el problema 25 con casi un 40% mejor que MIP en 15 segundos y 7,5% mejor que GRASP, con 5 minutos más de ejecución.

La penúltima columna muestra la distancia en aristas en común entre la solución óptima o límite encontrada por el solver MIP y la del AG, mientras que la relativa se presenta en la última. En el problema 3, podemos ver que la solución encontrada por ambas

metodologías es la misma y no se trata de una coincidencia de costos totales iguales. El otro caso para el que se tuvo una solución muy cercana en su diseño topológico fue el del problema 6, con 3 aristas diferentes; sin embargo el costo fijo de estas incide en el costo total, ya que hay una diferencia de casi el 7% entre las dos soluciones. En los últimos 4 problemas se ven los valores más altos de distancia, lo que se debe a topologías muy diferentes, con una cantidad aproximada de 20 arcos menos en las soluciones óptimas del AG

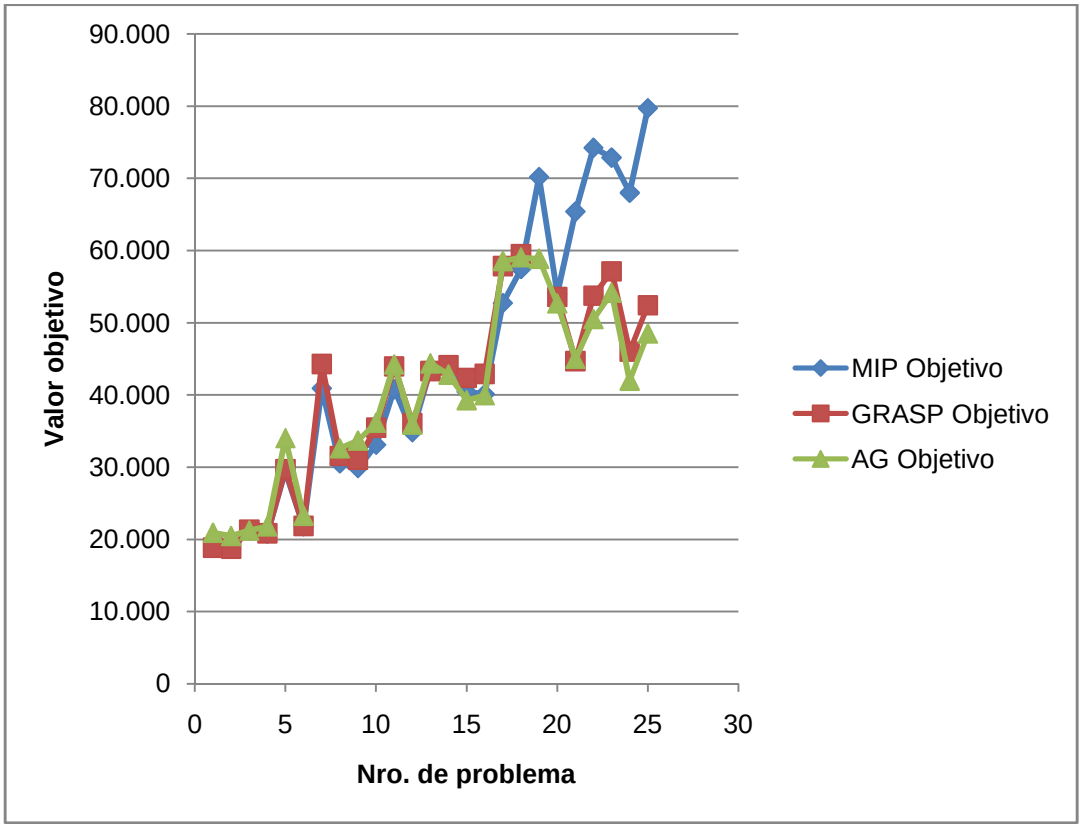


Figura 12: Comparación de valores objetivos entre los 3 métodos

La Figura 12 presenta un gráfico del mejor valor objetivo para cada método, en cada problema.

Otro tema a considerar es el del comportamiento de la evolución de los individuos a través de las distintas generaciones. Como ya se mencionó, el mejor valor de fitness para cada iteración se guardó en un vector, de manera de poder mostrar gráficamente el desarrollo de éste, algo de sumo interés para conocer como fue el comportamiento del AG

En la Figura 13 se puede ver la evolución del valor objetivo para el problema 3. En este caso se tiene una grafica bastante escalonada, estancándose en la mejor solución aproximadamente en la generación 19000. Puede notarse también que un poco antes, luego de un inmovilización entre 9213 y 15793, se produjo una mutación que trajo un individuo de muy buenas aptitudes, cercano en el espacio de soluciones al óptimo global, el cual aportó luego un linaje de hijos fuertes, fruto de varios cruzamientos, que desembocó en la salida óptima del problema.

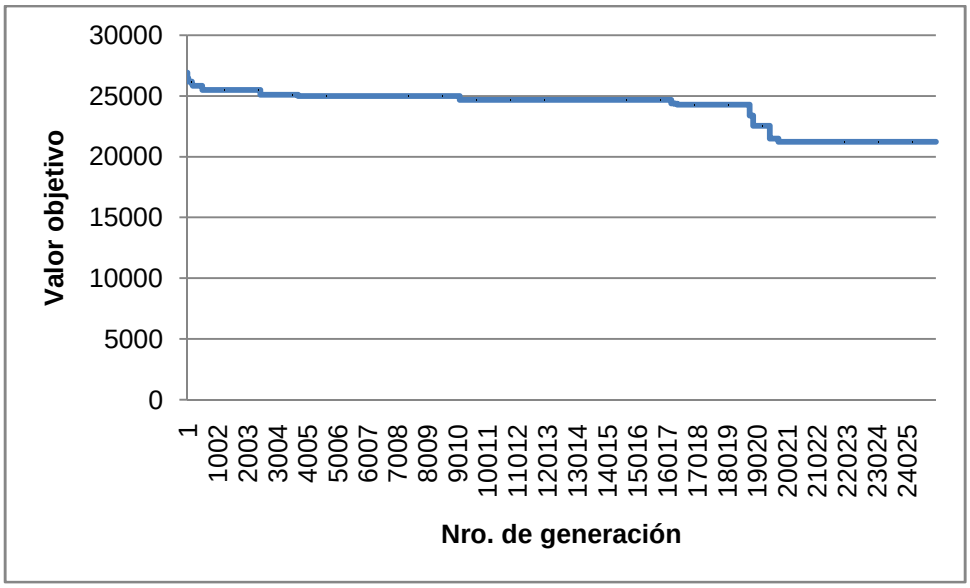


Figura 13: Evolución del problema 3

En la Figura 14 se presenta un ejemplo de convergencia prematura, con obtención de un óptimo local. Corresponde al problema 5, cuyo resultado para el AG presentó el mayor porcentaje de diferencia con la solución óptima del solver MIP. Puede notarse contra el eje vertical de la gráfica, una caída vertical hacia costos menores, causado por una población inicial generada aleatoriamente, con individuos sin mucha aptitud, pero que a razón de varios cruzamientos entre ellos, se producen en pocas generaciones linajes mejores.

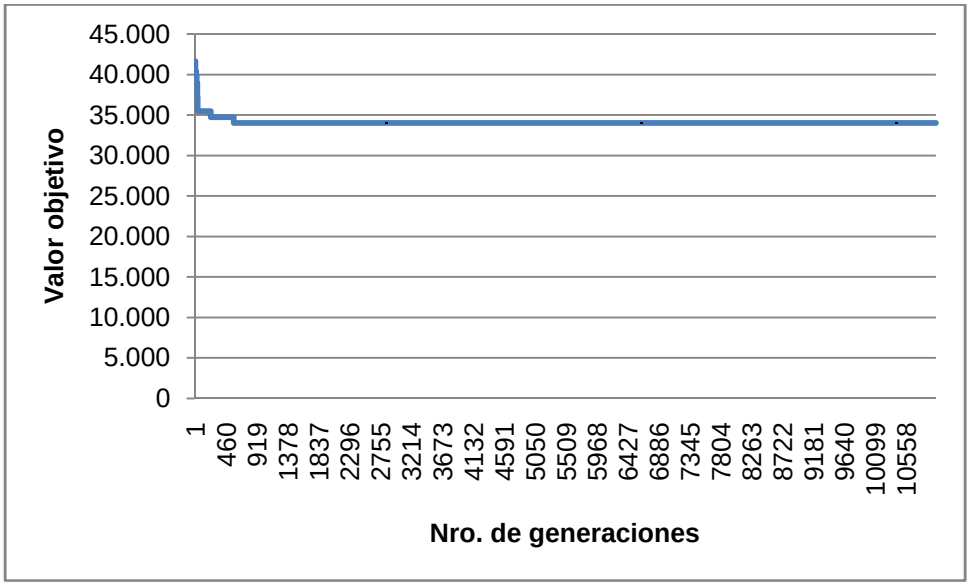


Figura 14: Evolución del problema 5

Luego, se produce el estancamiento y la mutación de un bit no logra traer mejora alguna desde la generación 580 aproximadamente en adelante. Esto puede dar a pensar la

presencia de individuos con esquemas de cromosomas muy parecidos entre sí, sin lograrse llegar a nuevos espacios de soluciones.

En la Figura 15 se ve un perfecto ejemplo de actuación de la mutación sobre la población. Corresponde al problema 20.

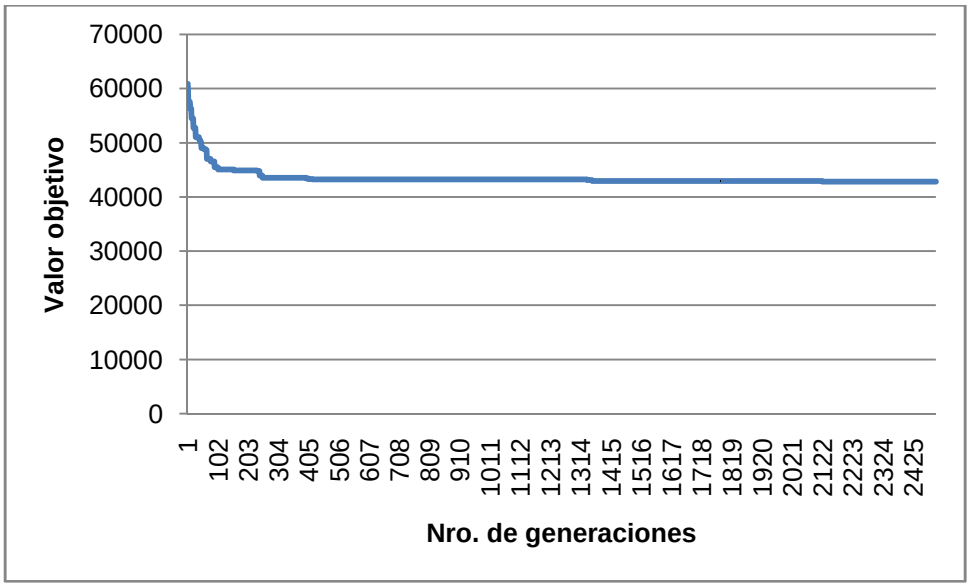


Figura 15: Evolución del problema 20

Como ocurrió con el problema 5, se da una caída vertical en los valores objetivos al inicio, para luego no darse ninguna mejora por varias iteraciones.

6 Conclusiones y trabajo futuro

Finalmente se cuenta con una aplicación de algoritmos genéticos, la cual devuelve resultados de forma rápida y efectiva para diseño de redes de telecomunicación en diferentes escenarios. La misma fue probada para 25 problemas, ejemplos de similares casos que se pueden dar en la realidad planteada del proyecto. Luego se contrastaron los resultados obtenidos para estos, con los de otras metodologías, como la del solver MIP y el algoritmo GRASP. El AG realizado para éste proyecto presentó un buen comportamiento, retornando un resultado, si no óptimo, bastante cercano al mismo (con no más de un 15% de diferencia); para un único caso, no se encontraron soluciones. Viendo la Tabla 3 presentada en el capítulo anterior, es lógico deducir, que a medida que el tamaño de los componentes del grafo aumenta (nodos y aristas), el AG deja resultados mejores que las otras dos metodologías, en un tiempo considerablemente menor. Esto deja la pauta de que se cuenta con una meta-heurística de muy buena performance en problemas bastante reales, redes “pesadas” con gran cantidad de arcos, problemas que con métodos de programación lineal tendrían resultados en tiempos exponenciales, mientras que aquí, se logran con menos de quince minutos. En comparación con GRASP, puede notarse por la Figura 12 que ambos tuvieron soluciones con valores óptimos muy cercanos, predominando el AG sobre los últimos cinco problemas, pero, nuevamente, con menos tiempo de resolución para éste último. Por otra parte, se notó que la eficiencia del AG merma en casos de poca holgura en la capacidad de los arcos, y mayores valores de demanda para los escenarios. Esto se debe seguramente a la heurística manejada para el ruteo del flujo, la cual descarta simplemente una solución si no se pueden enviar las demandas por los caminos escogidos, sin contar con formas para tratar de re-rutear el flujo que falló por falta de amplitud de banda. A pesar de esto, el AG falló únicamente para un problema de esa índole, mientras que en otros similares se tuvieron siempre resultados aceptables.

De todo esto, se puede entonces decir que al momento de querer buscar una amplia gama de buenas soluciones, que no tengan por qué ser las óptimas, en tiempos relativamente cortos, el AG desarrollado para el proyecto puede ser una buena opción. Y la afirmación previa cobra más fuerza si se trata de redes donde se tiene una cantidad considerable de nodos y links. También, los algoritmos genéticos cuentan con la ventaja de ser más flexibles a cambios en las condiciones, por lo tanto, se podría utilizar éste mismo AG para problemas similares de diseño topológico de redes, con otras restricciones diferentes; para esto alcanzaría con intercambiar los algoritmos complementarios que se manejaron, por otros que busquen la factibilidad de las soluciones de acuerdo a las nuevas condiciones. El desconocimiento por parte del cromosoma de la topología que está representando, trae como desventaja al AG desarrollado, que sean necesarios los algoritmos complementarios para evaluar si una solución es factible o no, lo que lleva a que cada candidato, resultante de una reproducción por cruzamiento y/o una mutación, deba ser sometido a todos los procedimientos necesarios para determinar su factibilidad, aunque al final, luego de probablemente ser modificado por estos, termine resultando en un individuo ya existente en la población.

Con respecto a las dificultades encontradas durante el desarrollo del proyecto, si bien cabe aclarar que existieron, fue una suerte que no se dieron en demasía. Durante el diseño, la principal dificultad vino de no contar con pruebas certeras de qué podría funcionar o que no. Se desconocía si el cruzamiento en un punto alcanzaría o sería necesario agregar un punto más de cruce (de hecho, al finalizar la primera versión del AG, el cruzamiento era en un punto, luego al ver que podían pasar generaciones sin haber creado un solo individuo nuevo, se pasó a 2). Al momento de escoger qué métodos complementarios utilizar para el cumplimiento de las restricciones, resultaba en más de un caso, difícil evaluar hasta donde servía lo recopilado por la documentación leída, si realmente satisfacía lo que se buscaba hacer o no. También se debió utilizar una selección por ruleta adaptada, manejando conceptos parecidos a la selección proporcional, debido a que la optimización del problema busca valores mínimos en lugar de máximos. Lo que se terminó realizando, fue darle más probabilidad de ser seleccionados para reproducción a individuos más aptos, saliendo esta probabilidad alta de los valores más grandes de fitness en proporción con el total, o sea, los costos de los individuos menos adaptados.

A modo de mejoras o extensiones a realizar para éste proyecto, se puede comenzar concentrando la atención a lo ocurrido con el problema para el cual no se obtuvieron resultados. Por razones de tiempo no se pudo experimentar lo suficiente con heurísticas alternativas de ruteo de flujo; hubiese estado interesante contar con una versión que maneje otra opción para esto. La mayor parte de las soluciones descartadas al probar el problema 7, eran consideradas como no factibles al momento del ruteo, ya que al primer intento de no poderse enviar una demanda por completo, la solución se descartaba. Quizás un método basado en *backtracking*, que se encargue de probar otra combinación de caminos para realizar un re-ruteo sea de mejor efectividad, o en una tarea más “pesada”, manejar conjuntos de caminos arista disjuntos para cada requerimiento, y comenzar a probar las distintas combinaciones de valores de flujo en los arcos de estos para cada requerimiento, hasta lograr el ruteo demandado.

También podría buscarse una metodología mejor que la de un algoritmo ávido, para el caso de encontrar los caminos arista disjuntos de cada demanda; se conoce la efectividad de estos al momento de buscar soluciones rápidas, pero raramente, esas son las más óptimas. Por otra parte, cabe aclarar que el algoritmo de Dijkstra implementado para los caminos de menor costo, se comporta de forma eficiente, retornando buenos caminos con bajos costos tanto fijos como variables.

Finalmente, puede resultar interesante experimentar con una fuerza selectiva alternativa, más controlada, que no proporcione más ventaja a los individuos excesivamente aptos. La modalidad usada puede traer problemas de convergencia prematura, una situación que probablemente sea la causa en más de una ejecución realizada, de que la evolución del mejor valor de fitness para cada generación se estanque luego de las primeras 600 iteraciones, obteniéndose gráficas similares a la de la Figura 14 para algunos problemas. Existen otras formas de selección, como por ejemplo: por torneo o por ranking (Gil Londoño, 2006), que podrían resultar interesantes de probar en el AG y que por razones de tiempo no se pudieron realizar.

Referencias

Abuali, F. N., Schoenefeld, D. A., & Wainwright, R. L. (1994). Terminal assignment in a communications network using genetic algorithms. *ACM Annual Computer Science Conference* (págs. 74-81). Phoenix, Arizona, United States: ISBN:0-89791-634-4.

Blessing, J. (2000). Efficient network design using heuristic and genetic algorithms. En *Telecommunications Optimization: Heuristic and Adaptive Techniques* (págs. 35-55). David W. Corne, Martin J. Oates, George D. Smith.

Bubble-Sort. (5 de enero de 2009). *Wikipedia*. Recuperado el 8 de enero de 2009, de http://es.wikipedia.org/wiki/Bubblesort

Coello Coello, C. A. (1995). Introducción a los Algoritmos Genéticos. *Soluciones avanzadas. Tecnologías de Información y Estrategias de Negocios No. 17* , 5-11.

Cox, A., Orvosh, D., Davis, L., & Qiu, Y. (1993). A Genetic Algorithm for Survivable Network Design. *Proceedings of the 5th International Conference on Genetic Algorithms* (págs. 408-415). ISBN:1-55860-299-2.

Dengiz, B., Altiparmak, F., & Smith, A. E. (1997). *Local Search Genetic Algorithm for Optimal Design of Reliable Networks.* IEE Transactions on evolutionary computation, vol. 1, No. 3.

Dijkstra, a. (30 de diciembre de 2008). *Wikipedia*. Recuperado el 3 de enero de 2009, de http://es.wikipedia.org/wiki/Algoritmo_de_Dijkstra

Gil Londoño, N. (2006). *Algoritmos Genéticos.* Medellín: Universidad Nacional de Colombia.

Ko, K.-T., Tang, K.-S., Chan, C.-Y., Man, K.-F., & Kwong, S. (1997). Using genetic algorithms to design mesh networks. *Computer, Volume 30* , 56-61.

Kumar, A., Pathak, R. M., Gupta, Y. P., & Parsaei, H. R. (1995). A genetic algorithm for distributed system topology design. En *Computers and Industrial Engineering, vol. 28* (págs. 659-670). ISSN:0360-8352.

Mersenne-Twister. (14 de marzo de 2009). *Wikipedia*. Recuperado el 22 de marzo de 2009, de Wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister

Nesmachnow, S., Cancela, H., & Alba, E. (2004). Técnicas evolutivas aplicadas al diseño de redes de comunicaciones confiables. *Actas del III congreso español sobre metaheurísticas, algoritmos evolutivos y bioinspirados.*

Olivera, A., Robledo, F., & Testuri, C. (December 15-17, 2008). Solving a capacitated, fixed charge, multicommodity network flow problem with uncertain demand and survivability constraints. *Proceedings of the VI ALIO/EURO Workshop on Applied Combinatorial Optimization.* Buenos Aires, Argentina.

Smith, A. E., & Dengiz, B. (2000). Evolutionary methods for the design of reliable networks. En *Telecommunications Optimization: Heuristic and Adaptive Techniques* (págs. 17-34). David W. Corne, Martin J. Oates, George D. Smith.

Anexo - Manual de Usuario para el Ejecutable

La implementación fue realizada con DEV-C++ v4.9.9.2 como IDE y compilador para C++, en un sistema operativo Windows Vista, dando como resultado un programa ejecutable por consola. Esta sección consiste en una breve descripción de cómo funciona éste programa, ubicación de los archivos de configuración de entrada y los de salida con los resultados, invocación del ejecutable, etc.

El programa ejecutable se llama AG.exe

Por línea de comando recibe como parámetro el archivo de configuración con el grafo inicial, las demandas y los escenarios. No hay restricciones con el tipo de extensión del mismo, siempre que sea de lectura. En el capítulo 4 de éste documento, sección 4.1.4 Serializer, se da una descripción de la sintaxis del archivo.

Ejemplo de invocación del ejecutable en una consola de Windows: "*C:\AG.exe problema.txt*"

Inmediatamente luego de iniciado el programa, se muestra la semilla aleatoria obtenida para la ejecución y se solicita el ingreso del tamaño de la población deseada (la cantidad mínima de población soportada es de 3 individuos).

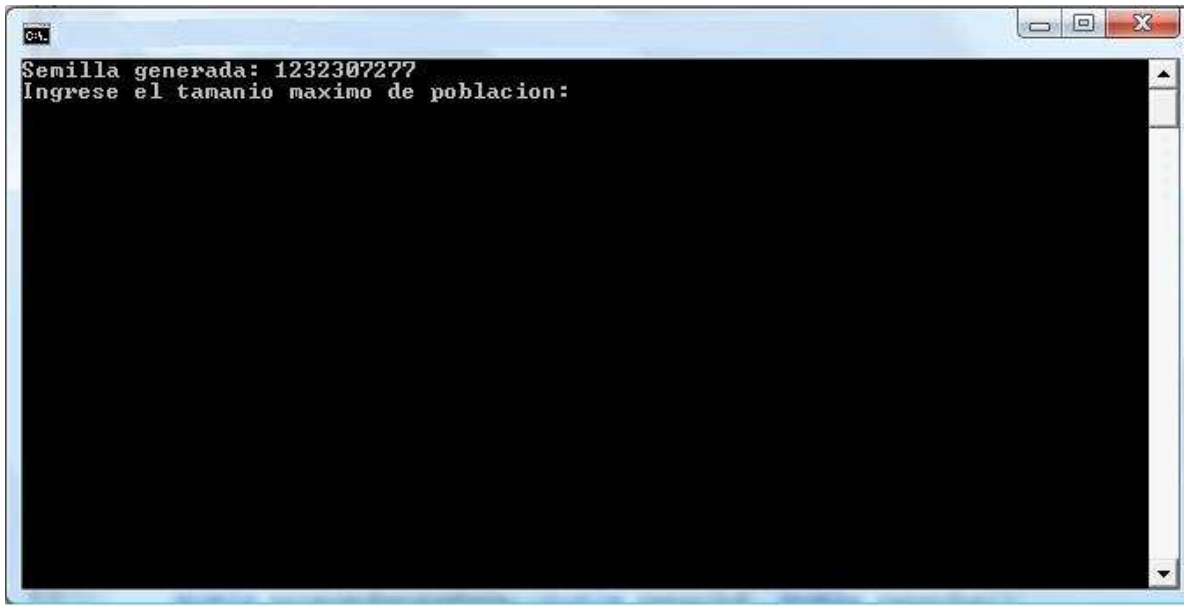


Figura 16: Pantalla de inicio de la ejecución

Luego de ingresado el tamaño de población, se solicita la cantidad máxima de generaciones para el algoritmo genético (Figura 17). Después de obtenidos estos dos parámetros, el algoritmo genético comienza su ejecución, mostrándose un mensaje al usuario de que aguarde hasta alcanzarse el número de generaciones deseadas. Un segundo o dos luego de empezada la ejecución, se muestra la cantidad de población inicial obtenida, dato que puede servir como guía del tipo de problema que se está probando, podría darse que se haya encontrado un tamaño de población menor que el máximo indicado en el caso de problemas "pesados".

Figura 17: Ingreso de la cantidad de generaciones

Figura 18: Pantalla durante el algoritmo en ejecución

Una vez finalizado el algoritmo, el programa pregunta si se desean calcular, y ver por pantalla, los valores de distancia en cantidad de aristas entre la solución óptima encontrada y la del solver MIP En caso de que se desee calcular estos datos, es necesario de que exista en el mismo directorio donde se encuentra el ejecutable, un archivo de nombre “solucionmip.txt” conteniendo solamente los pares de nodos que conforman las aristas para la solución óptima hallada con MIP, separados por un tabulador, y cada par ocupando únicamente un renglón (en la descripción del módulo

Serializer del capítulo 4 sección 4.1.4, se presenta también una descripción de la sintaxis para éste archivo).

Figura 19: Consulta al usuario por el cálculo de valores de distancia

En caso de que se opte por no desear ver estos resultados, la ejecución finaliza, mostrándose un mensaje que confirma la correcta conclusión del algoritmo, en caso contrario, se presentan los valores de distancia calculados, como muestra la Figura 20, para luego pasar a culminar la ejecución.

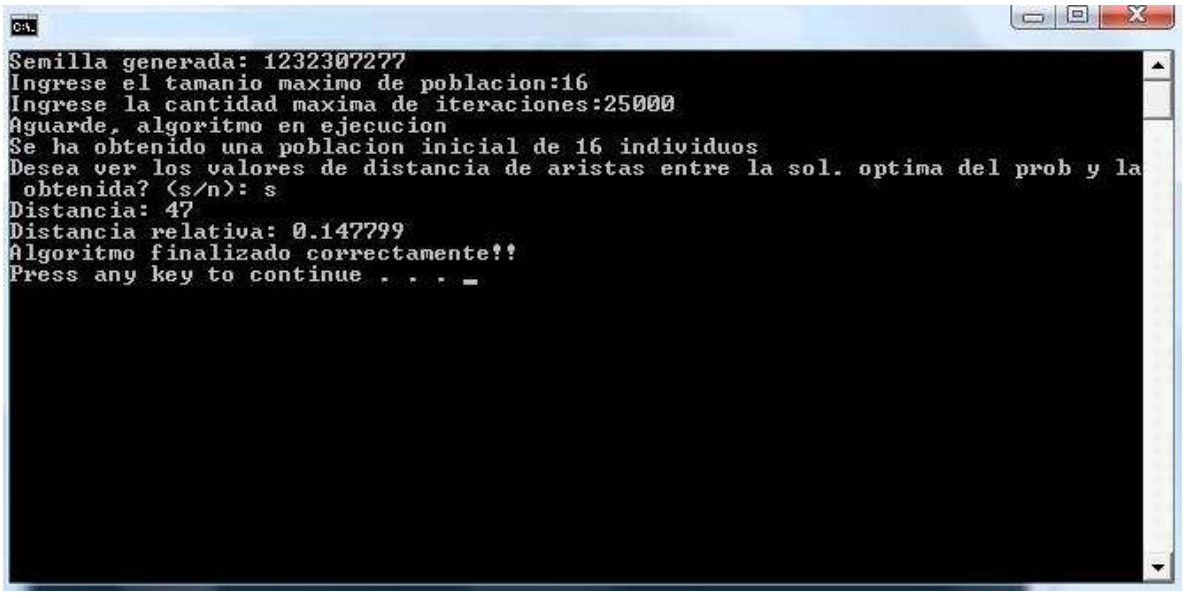


Figura 20: Culminación exitosa, con datos desplegados por pantalla

El ejecutable da como salida dos archivos, situados en el mismo directorio del ejecutable:

1. Salida.txt: con la topología para la mejor solución encontrada, junto con su costo, y el tiempo que llevó la ejecución; más datos presentes en éste archivo son nombrados en el capítulo 4 sección 4.1.4 Serializer.
2. Salida_evolucion.txt: muestra para cada generación, el valor de la mejor solución existente entre la población