Optimización y Diseño de Redes Diámetro-Confiables

Optimización y Diseño de Redes Diámetro-Confiables

Proyecto de Grado / Ing. en Computación

Departamento de Investigación Operativa
Instituto de Computación - Facultad de Ingeniería
Universidad de la República (UDELAR)
Montevideo - Uruguay

Julio de 2002

Pablo Burgos - Alfredo Godoy

Tutores

Dr. Ing. Héctor Cancela
Facultad de Ingeniería - Universidad de la República (UDELAR)
Departamento de Investigación Operativa - Instituto de Computación
Montevideo - Uruguay

Dr. Ing. Luis Petingi
City University of New York (CUNY).
Nueva York, E.U.A.

Resumen

Centrado en un nuevo problema de optimización combinatoria complejo dado a conocer recientemente, este trabajo resulta ser el primero en su clase en estudiar el Diseño y Optimización de Redes Diámetro-Confiables.

En el transcurso del mismo se ha analizado detalladamente un variado número de metaheurísticas, determinando las dos más adecuadas para el problema en cuestión. Constituyendo este trabajo el primero en su clase, resultan originales todas las ideas en las que se basan los algoritmos desarrollados, sin considerar las básicas y generales sugeridas por cada metaheurística.

El resultado final constituye la presentación de los dos primeros algoritmos que diseñan y optimizan redes según la Diámetro-Confiabilidad. Los mismos fueron desarrollados sobre la base de los principios promulgados por Algoritmos Genéticos y GRASP. Se presenta asimismo un algoritmo polinomial para simplificar, bajo ciertas condiciones, una red a otra equivalente según la Diámetro-Confiabilidad. Los algoritmos en su totalidad exhiben en la práctica un muy buen desempeño para los más de 500 juegos de pruebas realizados.

Palabras clave:

Confiabilidad en Redes, Diámetro-Confiabilidad, Metaheurística, GRASP, Algoritmos Genéticos.

Keywords:

Network Reliability, Diameter Constrained Reliability, Metaheuristic, GRASP, Genetic Algorithms.

Capítulo 1
Introducción

1. Introducción

Este taller se enmarca en la línea de trabajo sobre el diseño de redes de comunicaciones confiables, que ha incluido desde 1990 y hasta el presente los proyectos "*Algorítmica no numérica y Evaluación de la confiabilidad en Redes*", "*Modelización y Simulación de Sistemas Complejos en Ambientes Inteligentes: Herramienta de Diseño*" – Proyecto BID-CONICYT 153/92-, "*Evaluación de la seguridad de funcionamiento*", y "*Diameter constrained network reliability: theory and applications*". Estos proyectos han recibido financiación del PEDECIBA Informática, del CONICYT, del INRIA- Francia, y de la City University of New York.

En el marco del mismo se esta desarrollando algoritmia para el cálculo de la confiabilidad de redes y para la optimización de éstas (herramienta HEIDI).

El proyecto que se ha asignado a éste, nuestro taller, es en esencia una extensión al proyecto "*Diameter constrained network reliability: theory and applications*". El impulso que motiva esta extensión, esta actualmente siendo llevado adelante por el Dr. Ing. Héctor Cancela y el Dr. Ing. Luis Petingi y se centra en el problema del cálculo de la Diámetro-Confiabilidad de una red; problema presentado por ellos, el cual constituye un problema de optimización combinatoria complejo. Este problema es nuevo en esencia, no existiendo trabajo práctico alguno al respecto. El mismo puede formularse informalmente de la siguiente manera (una formalización del problema es presentada en la Sección 3.2 *Confiabilidad en redes*):

> **Problema objetivo de nuestro taller:**
> Considérese una red definida como un conjunto de componentes y sus interconexiones, de los que se conoce su costo y confiabilidad, y sean s y t dos de los componentes de dicha red, teniéndose una restricción de distancia D y una restricción de costo K. En estas condiciones se debe determinar la topología de la subred que tenga un costo total no mayor al parámetro K, y que maximice la Diámetro-Confiabilidad entre s y t sujeta a la restricción de distancia D. La Diámetro-Confiabilidad entre s y t sujeta a la restricción D, puede verse como la probabilidad de que frente a fallas en los componentes de la red, los componentes s y t estén conectados por algún camino de largo a lo sumo D.

Los problemas de optimización en general surgen naturalmente en diferentes campos de la actividad humana como por ejemplo en las ciencias y la industria. Entre éstos se encuentran los problemas de optimización combinatoria, como es el caso de nuestro problema objetivo en este taller, en los que a partir de un conjunto finito de soluciones factibles para un problema dado y de una función definida sobre ese conjunto, se pretende determinar la solución, perteneciente a ese conjunto de soluciones, que minimiza o maximiza la función objetivo para el problema en cuestión. Sin embargo, a pesar de que el número de soluciones factibles está acotado, suele suceder que el mismo crece exponencialmente con el tamaño del problema, dificultando enormemente, e imposibilitando en la práctica, el llevar a cabo un análisis exhaustivo de cada una de las soluciones factibles para problemas de determinado porte. Esta limitante en el tamaño está también presente en el problema que nos concierne.

Para hacer frente a esta limitación se han desarrollado estrategias que permiten determinar una solución suficientemente buena, sin que sea necesario el análisis de todas las soluciones factibles; obteniéndose por esta vía algoritmos “no exactos”, en el sentido de que no necesariamente obtienen la solución óptima, pero suficientemente adecuados en lo que refiere al fin práctico para el que han sido diseñados.

Este enfoque para determinar la solución óptima, o una suficientemente buena, encaja perfectamente con el promulgado por las metaheurísticas en general, las que históricamente han demostrado ser extremadamente útiles en estos casos. Cada metaheurística proporciona un modelo a partir del cual es posible desarrollar algoritmos que implementan una estrategia particular de búsqueda de la mejor solución. En el transcurso de este taller hemos analizado un variado número de metaheurísticas determinando las dos más adecuadas para el problema en cuestión. Dicha selección de las dos metaheurísticas más adecuadas se basó en una fina comparación entre las características de cada una, y de las bondades de cada característica frente a la naturaleza del problema objetivo. Posteriormente hemos desarrollado e implementado algoritmos basados en las que resultaron seleccionadas. Siendo este trabajo, el primer de su clase dedicado al análisis, selección y posterior implementación de las metaheurísticas seleccionadas, resultan originales todas las ideas en las que se basan los algoritmos desarrollados, sin considerar las básicas y generales sugeridas por cada metaheurística.

Puntualizamos que la información contenida en este documento está estructurada en varios Capítulos. En el Capítulo 2 se incluyen los Objetivos planteados para el presente trabajo. En el Capítulo 3 se describe un marco conceptual necesario. Una breve presentación de las metaheurísticas analizadas, así como la definición de los criterios de selección y el proceso de selección propuestos se encuentran en el Capítulo 4. Los Capítulos 5 y 6 presentan respectivamente los algoritmos desarrollados y los resultados experimentales. El Capitulo 7 exhibe las conclusiones finales. Las publicaciones y trabajos referenciados se citan en el Capítulo 8. Finalmente, el Capítulo 9 incluye material anexo a este documento.

Finalizando esta sección indicamos que debido a que este taller constituye la primera experiencia práctica aplicada al Diseño de Redes Diámetro-Confiables, aporta al campo de la confiabilidad en redes los primeros resultados empíricos para este problema, habiéndose obtenido resultados superiores a lo que era de presumirse desde un principio.

Capítulo 2
Objetivos

2. Objetivos

Los objetivos estipulados al comienzo del trabajo definieron en gran parte las actividades que se desarrollarían y los productos que finalmente se habrían de entregar.

Cada objetivo estipulado implica un aporte en un determinado sentido. Presentamos a continuación aquellos referidos a este trabajo:

2.1. Aportes Solicitados para este Taller

- Proporcionar un análisis de las metaheurísticas más utilizadas respecto de su aplicabilidad al Diseño de Redes Diámetro-Confiables, problema para el que actualmente existe escasa información tanto en lo referente al problema en sí como a datos de trabajos empíricos sobre el mismo. Proporcionar una indicación de cuáles son las dos más adecuadas para atacar dicho problema.
- Proporcionar algoritmos que pongan en práctica los principios de las dos metaheurísticas más idóneas para este problema. Estos estarán implementados en el lenguaje de programación C++, garantizando mayores posibilidades de portabilidad y reutilización.
- Proporcionar datos experimentales sobre el desempeño de los algoritmos implementados y un análisis de los mismos, los que avalarán los resultados obtenidos y asistirán a las futuras investigaciones entorno al Diseño de Redes Diámetro-Confiables.
- Asegurar la integración con HEIDI.

Capítulo 3
Marco Conceptual

3. Marco Conceptual

3.1. Metaheurísticas

Al intentar resolver problemas de optimización suelen utilizarse métodos iterativos como base de los algoritmos que finalmente se implementan y utilizan con tal fin. La eficiencia de estos métodos iterativos depende principalmente del modelo usado, el cual define la estrategia de búsqueda de la solución al problema. Hay que tener en cuenta, sin embargo, que un fino ajuste de los parámetros que controlan el desempeño de los algoritmos implementados no mejorará una mala elección de la estructura de la vecindad o de la función objetivo utilizadas para encontrar la solución óptima; siendo por esto que, en un modelo efectivo prima el uso de técnicas de búsqueda robustas, las que por lo general no atribuyen demasiado peso a los diferentes parámetros que las controlan.

Al analizar estos métodos iterativos es útil pensar que éstos, en su búsqueda de la solución óptima, llevan a cabo una recorrida sobre el grafo de espacio de soluciones asociado al problema objetivo. Este grafo de espacio de soluciones puede definirse de la siguiente manera:

Sea G=(S,A) la representación del grafo de espacio de soluciones asociado a un problema, entonces se tiene que:

- El conjunto de vértices S se corresponde con el conjunto de soluciones factibles, es decir, cada vértice del grafo representa una solución factible para el problema.
- El conjunto de aristas dirigidas A es tal que, una arista de vértices i y j pertenece a A si j pertenece a la vecindad de i. Una cierta solución factible j pertenece a la vecindad de i si a partir de la solución factible i aplicando la definición propuesta de vecindad se obtiene o se llega a la solución factible j. Donde la vecindad es una relación definida de forma particular al problema y al algoritmo con el que se quiere resolver dicho problema.

Elegir una solución inicial para el método iterativo puede verse entonces como seleccionar un vértice en el grafo de espacio de soluciones G, vértice a partir del cual, se comenzará la recorrida sobre el grafo. Teniendo esto en mente, es posible interpretar que con cada ciclo, el procedimiento iterativo se mueve sobre el grafo desde el vértice actual hacia un vértice adyacente; donde el objetivo es poder alcanzar en un número finito y acotado de movidas el vértice que representa la solución factible óptima del problema a resolver.

Los algoritmos iterativos, a lo largo de su búsqueda, no necesariamente se mantienen siempre dentro del espacio factible de soluciones, sino que pueden llevar a cabo movidas que los lleven a visitar soluciones que cumplen solamente un subconjunto de restricciones del problema. Esto flexibiliza la definición de grafo de espacio de soluciones dada anteriormente, dando lugar a un grafo menos restrictivo denominado grafo de espacio de estados, donde cada vértice de éste representará entonces una solución pasible de ser visitada por el algoritmo a lo largo de su búsqueda, siendo ésta una solución factible o no para el problema.

Dado un problema de optimización, existen usualmente muchas formas de definir el grafo de espacio de estados G=(S,A), sin embargo la idea esencial consiste en elegir uno que satisfaga la siguiente condición:

> Dado un vértice i en S, debe existir en G un camino desde i hacia la solución óptima i*.

Cabe remarcar que la elección de cada algoritmo debe ser realizada cuidadosamente para evitar dejar parte del espacio de estados sin visitar, ya que en este espacio no visitado puede que se encuentre la solución óptima.

En la resolución de problemas de optimización suele ser impracticable el uso de algoritmos que lleven a cabo una recorrida exhaustiva sobre el grafo de espacio de estados, principalmente porque esta búsqueda exhaustiva requeriría tiempo y recursos que dependen en forma exponencial del "tamaño" del problema original. Debido a esto, se utilizan algoritmos que recorren este grafo utilizando una cierta filosofía con la que pretenden alcanzar la solución óptima evitando una recorrida completa del mismo. Como ejemplo de algoritmos que utilizan filosofías de este tipo al llevar a cabo la recorrida sobre el espacio de estados, es posible citar a los algoritmos desarrollados sobre la base de metaheurísticas.

Dado que en este taller se desarrollarán algoritmos sobre la base de metaheurísticas, corresponde tener en cuenta que el éxito de una metaheurística depende de su capacidad de:

- Adaptarse a casos particulares del problema
- Evitar quedar presa en óptimos locales
- Explorar en buena forma la estructura del espacio de soluciones del problema.

Además se debe tener en cuenta, en qué grado la metaheurística permite:

- El uso de procedimientos de reinicialización.
- El uso de aleatoriedad controlada, la que en general es aplicada a la diversificación en la búsqueda.
- El uso de eficientes estructura de datos.
- El pre-procesamiento del problema original.
- La búsqueda intensa en regiones promisorias del espacio de soluciones.
- El explorar al máximo las posibilidades de mejora de una solución en particular.

Observaciones como las anteriores, ocasionaron la búsqueda de diferentes maneras de atacar los problemas de optimización, y fueron la base del desarrollo de muchas metaheurísticas que, de forma comprobada, mejoraran la capacidad de obtención de soluciones para problemas difíciles; como por ejemplo:

- Simulated Annealing
- Algoritmos Genéticos
- Tabu Search
- GRASP

Más adelante en este trabajo se analizarán varias metaheurísticas describiéndose con cierto grado de detalle cada una de ellas y analizando su aplicación sobre el problema de Diseño de Redes Diámetro-Confiables. De todos modos, a continuación presentamos una breve reseña sobre las metaheurísticas que se discutirán posteriormente:

- Algoritmos Genéticos - [Holland 1975 [H1975]; Goldberg 1989[G1989a]]
  Metaheurística basada en los modelos evolutivos aplicables a las especies vivas, modelos evolutivos desarrollados originalmente en el dominio de las Ciencias Naturales.

- Ant Systems - [Colorni-Dorigo-Maniezzo 1991 [CDM+1991]]
  El desarrollo de esta metaheurística se basó en la observación del comportamiento de las colonias de hormigas reales, en particular en la manera en la que una colonia de hormigas resuelve el problema de encontrar el alimento y transportarlo hacia el hormiguero utilizando los caminos más cortos posibles.

- GRASP – [Feo-Resende 1989 [FR1989]]
  Esta metaheurística utiliza una mecánica de búsqueda en el espacio de soluciones que se basa en repetir iterativamente la siguiente secuencia de tareas: generar mediante un proceso aleatorizado una solución factible perteneciente al grafo de espacio de soluciones (esto define la fase de construcción); mejorar todo lo que sea posible la solución hallada en la fase de construcción, esto se hace en la fase de búsqueda local; comparar la mejor solución encontrada hasta el momento con la solución provista por la fase de búsqueda local y almacenar a la mejor de ellas como la mejor solución encontrada hasta el momento. Al final de cada ciclo se suelen ajustar de forma automática algunos parámetros que adaptarán la búsqueda de la solución en las siguientes fases.

- Neural Nets [Hopfield-Tank 1985 [HT1985]]
  La base de esta metaheurística se encuentra en los modelos desarrollados para describir el funcionamiento del sistema nervioso en general y del cerebro humano en particular. Su idea esencial es la de utilizar un buen número de entidades procesadoras de información (llamadas neuronas), de modo que al trabajar en conjunto logren obtener la solución al problema que se desea resolver.

- Simulated Annealing - [Van Laarhoven-Aarts 1987]
  A lo largo del tiempo la humanidad ha desarrollado diferentes técnicas para trabajar con los metales; técnicas que se han perfeccionado a lo largo de las generaciones. Esta metaheurística se basa en una moderna técnica utilizada en la producción de metales con estructura cristalina perfecta o cuasiperfecta, y lleva a cabo una filosofía de trabajo que imita los pasos usados al aplicar esta técnica.

- Tabu Search - [Glover 1989-1990 [G1989b] [G1990]]
  Ciertos análisis sobre los métodos de recorrida empleados en el espacio de soluciones por parte de los diferentes algoritmos utilizados en problemas de optimización, han revelado ciertas debilidades y dificultades que obstaculizan el descubrimiento de la solución óptima. A partir de estos análisis se han desarrollado estrategias para evitar estos obstáculos mientras se intenta alcanzar la solución óptima. Entre éstas se encuentra la metaheurística Tabu Search, la que propone, entre otras cosas, mantener una cierta historia de las visitas realizadas en dicho espacio de soluciones.

3.2. Confiabilidad en Redes

En esta sección introducimos los conceptos básicos necesarios para el entendimiento de este documento y de nuestro trabajo en general.

Grafo: Un grafo $G=(X, U, \psi)$ es una terna constituida por:

1- Un conjunto $X=\{x_1, x_2, ...,x_n\}$ finito, numerable y no vacío (existen grafos infinitos).

2- Una familia $U=\{u_1, u_2, ...,u_m\}$, donde $u_i = [x_i, x_j]$ son pares de elementos tomados del conjunto X.

3- La función de incidencia ψ que asigna un par de vértices extremos a cada una de las aristas de G, $\psi(u)=[x, y]$

Un grafo es orientado cuando existe una relación de precedencia entre los elementos de una arista. En estos casos U es una familia de pares ordenados resultante del producto cartesiano de X.

Orden:

Dado un grafo $G=(X, U, \psi)$ se define el orden de G, Orden(G), como el cardinal del conjunto X.

Autoarista:

Dado un grafo no dirigido $G=(X, U, \psi)$, una arista $u \in U$ es una autoarista sii es de la forma [x, x], con $x \in X$.

Nodo adyacente a x:

Dados un grafo no dirigido $G=(X, U, \psi)$ y vértices x e y de G, se dice que y es adyacente a x sii existe una arista $u \in U$ de la forma [x, y].

Grado de x:

Dados un grafo no dirigido $G=(X, U, \psi)$ y un vértice x de G, se define Grado(x) como el número de aristas que tienen como uno de sus elementos el vértice x.

Camino:

Dados un grafo no dirigido $G=(X, U, \psi)$ y vértices $x_1, x_2,..., x_{n-1}, x_n$ de G, la secuencia de vértices $x_1, x_2,..., x_{n-1}, x_n$ es un camino sii $[x_1, x_2],..., [x_{n-1}, x_n]$ son aristas de G.

Camino Simple:

Dados un grafo no dirigido G=(X, U, ψ) y un camino c de G, se dice que c es un camino simple sii no repite vértices.

Largo de un Camino:

Dados un grafo no dirigido G=(X, U, ψ) y un camino c de G, se define largo(c) como el número de aristas que lo definen.

Distancia entre vértices:

Dados un grafo no dirigido G=(X, U, ψ) y vértices x e y de G, se define la distancia entre x e y como:

d(x, y)=min {largo(c), con c camino simple en G de la forma $x,x_2,...,x_{n-1}$, y}

Grafo Completo:

Dado un grafo no dirigido G=(X, U, ψ), se dice que G es un grafo completo sii $\forall$ x, y $\in$ X, x $\neq$ y se cumple que x es adyacente a y.

Subgrafo:

Dado los grafos no dirigidos G=(X, U, ψ) y H=(X', U', ψ'), se dice que H es un subgrafo de G sii se cumplen:

1- X' $\subseteq$ X

2- U' $\subseteq$ U

3- $\forall$ u' $\in$ U' se cumple ψ'(u')= ψ(u')

K-Diámetro de un grafo G:

Dado un grafo no dirigido G=(X, U, ψ), y un conjunto K $\subseteq$ X, se define K-Diámetro de G a la distancia máxima entre cualquier par de vértices de K.

K-Diámetro confiabilidad de un grafo G sujeta a una restricción de diámetro:

Dado un grafo no dirigido G=(X, U, ψ), un conjunto K $\subseteq$ X, y un valor real D, se define la K-Diámetro confiabilidad de G sujeta a una restricción de diámetro D como la probabilidad de que frente a fallas en los componentes de la red, las aristas sobrevivientes generen un subgrafo cuyo K-Diámetro no exceda el valor D. Se denota como R_K(G, D).

Diámetro-Confiabilidad entre s y t sujeta a la restricción D:

Se define como la K-Diámetro confiabilidad de G sujeta a una restricción de diámetro D cuando K={s, t}. Se denota como R_{st}(G, D).

Formalización del problema objeto de nuestro taller:

Dado un grafo no dirigido G=(X, U, ψ), un conjunto K={s,t} $\subseteq$ X, y un valor real D:

se quiere

$$\max_{U^* \subseteq U} R_{st}\,(G(X, U^*, \psi), D)$$

Sujeto a:

$$\Sigma_{u \in U^*}\ costo(u) <= K$$

Se asume que para cada elemento u $\in$ U se tiene definido el valor costo(u).

3.3. Principales Herramientas Utilizadas

En casi todo tipo de trabajos se hace necesario el uso de determinadas herramientas para obtener resultados de buena calidad o para facilitar la tarea, este taller no ha sido la excepción a esta regla. En esta sección puntualizamos brevemente algunas características de las herramientas más relevantes utilizadas, proporcionando así un marco que de alguna manera refleja el entorno básico de trabajo utilizado.

La primer herramienta a la que hacemos mención es al lenguaje de programación utilizado. En nuestro caso la implementación de los algoritmos fue llevada a cabo con C++.

Dado que C++ es un lenguaje ampliamente conocido nos limitamos a puntualizar las características que consideramos más relevantes en lo que se refiere a este taller:

- Soporta varios estilos de programación, entre los cuales se destacan: programación procedural, programación orientada a objetos y programación genérica.
- Es un lenguaje de propósito general, lo cual lo hace aplicable a diversos tipos de proyectos de desarrollo; en particular al nuestro. Esto le confiere además una muy buena expresividad al no ser un lenguaje diseñado para determinadas aplicaciones puntuales.
- No requiera ambientes de programación sofisticados.
- Es flexible y eficiente.
- Dispone de características que facilitan el desarrollo de programas fáciles de entender y de mantener, en particular por la habilidad de permitir el uso de abstracciones.

La otra herramienta a la que haremos referencia es a HEIDI. En la definición de casos de prueba es crucial disponer de juegos de datos correctos, en el sentido de que estos representen fielmente los casos de prueba que se quieren utilizar. Por esta razón, una herramienta que asista en la generación de casos de prueba facilitando tanto la definición de los mismos como la verificación de su correctitud, es invalorable; más aún, si provee una manera de llevar a cabo en forma intuitiva tanto la fase de generación como la de validación, características que se encuentran presentes en la herramienta HEIDI.

HEIDI es una herramienta desarrollada por el INCO (INstituto de COmputación) en el año 1992 en el marco del proyecto *BID/CONICYT 153/92 ``Herramienta de diseño de redes de comunicación confiables"* (con apoyo del PEDECIBA Informática y del Programa ECOS de cooperación científica entre Francia y Uruguay).

Su objetivo es el de asistir en la edición de redes desde el punto de vista de su topología, permitiendo asignar atributos a los componentes de esta y llevar a cabo el cálculo de ciertas características de la red diseñada.

Por esta razón es la herramienta ideal para generar las redes utilizadas en los casos de prueba necesarios. A continuación puntualizamos los aspectos más relevantes de esta herramienta en lo que respecta a este taller:

- Desarrollada para la plataforma Sun OS.
- Permite editar la topología de la red, y tanto exportarla hacia archivos como importarla desde archivos previamente exportados. Esta funcionalidad será la que utilizaremos para generar los casos de prueba que luego cargaremos y utilizaremos desde nuestros algoritmos.
- Permite asignar valores para determinados atributos de los nodos o aristas que constituyen la red. Entre estos atributos encontramos los siguientes: etiqueta, costo, confiabilidad y capacidad.
- Se permiten además funcionalidades para hacer zoom o cambio de escala, importantes en caso de que la topología de la red en cuestión resulte demasiado intrincada.

Capítulo 4
Análisis y Selección de Metaheurísticas

4. Análisis y Selección de Metaheurísticas

Esta sección la dedicamos a la presentación de cada una de la metaheurísticas analizadas; los criterios de selección que definimos y que posteriormente utilizamos al seleccionar las más adecuadas; y finalmente el resultado de esta selección.

Se estructura en tres subsecciones. La primera de las cuales está dedicada enteramente a la descripción de las características más relevantes de las metaheurísticas analizadas. En la segunda subsección presentamos la discusión llevada a cabo para determinar los criterios de selección así como también una presentación de los criterios en cuestión. El contenido de ésta, aunque algo breve en redacción, resume lo obtenido luego de las múltiples discusiones y decisiones que al respecto debimos tomar y es un punto crucial sobre el que se basará la posterior selección de las metaheurísticas a implementar, es un punto clave que condiciona los resultados de este taller y a esto se debe la importancia que le asignamos. En la tercer subsección se aplican los criterios definidos y se determinan las dos metaheurísticas más adecuadas para el problema objetivo del presente taller.

Brevemente, antes de comenzar con la presentación de las metaheurísticas, vale la pena puntualizar que éstas, en términos generales, se caracterizan por:

- Utilizar un cierto número de ensayos repetidos.
- Emplear uno o más agentes (neuronas, cromosomas, hormigas).
- Operar con un mecanismo de competición-cooperación en el caso en que se utilice más de un agente.
- Poseer procedimientos embebidos de automodificación de los parámetros de la metaheurística o de la representación del problema.

4.1. Metaheurísticas Analizadas

4.1.1. Ant Systems

Los insectos sociales han capturado la atención de los científicos principalmente por la alta estructuración de sus colonias, especialmente cuando estas se comparan con la simplicidad relativa de cada uno de sus individuos.

Dorigo y colegas fueron quienes primero propusieron los algoritmos Ant como una manera de hacer frente a los difíciles problemas de optimización combinatoria.

La inspiración para estos algoritmos Ant se originó en la observación llevada a cabo sobre colonias de hormigas reales, en las que el comportamiento mostrado esta dirigido más a la sobrevivencia del hormiguero como un todo que al de las hormigas simples.

Estos algoritmos Ant se basan en el destacable comportamiento utilizado por las hormigas para encontrar alimento, y en particular en la forma en que éstas descubren los caminos más cortos entre el hormiguero y el lugar donde se encuentra el mismo.

La clave para el descubrimiento de estos caminos más cortos radica en el manejo que las hormigas llevan a cabo de una sustancia química a la que se denomina feromona. Esta sustancia es depositada por las hormigas mientras caminan desde el hormiguero al alimento y viceversa, al hacerlo cada hormiga forma un rastro de feromonas que marca el camino que ha recorrido. A su vez, al elegir su camino, cada hormiga tiende a seleccionar los caminos que están marcados con las concentraciones más fuertes de feromona; de esta manera una hormiga puede llegar a las fuentes de alimento descubiertas por otras hormigas así como encontrar el camino de vuelta hacia su hormiguero a partir del lugar donde se encuentra el alimento. A lo largo del tiempo los caminos más cortos comienzan a ser transitados por la mayoría de las hormigas. Esto se debe a que las hormigas que los utilizan llegan antes a su destino y esto ocasiona que por los mismos circulen más hormigas por unidad de tiempo, por esta razón en estos caminos comienza a depositarse la mayor concentración de feromona y por tanto al transcurrir el tiempo todas las hormigas tienden a utilizarlos. Lo destacable es que este descubrimiento de los caminos más cortos se lleva a cabo cuando en conjunto las hormigas emplean (al caminar) este mecanismo guiado por los rastros de feromona.

En este mecanismo cada hormiga, a pesar de que puede construir una solución propia (un camino hacia la comida), proporciona un pequeño aporte en la construcción de la solución global encontrada por todas las hormigas en conjunto, solución global consistente en los caminos más cortos entre el hormiguero y el lugar en el que se encuentra el alimento. De esta manera es el conjunto de hormigas quien descubre esta forma óptima para llegar al alimento y regresar al hormiguero.

Otro hecho que tiene lugar durante este proceso es el comportamiento auto catalítico de las hormigas reales, es decir, existe tanto una retroalimentación positiva entre hormigas como una evaluación implícita de soluciones. Esto implica que al simular hormigas reales mediante agentes artificiales deberá tenerse algún mecanismo para evitar la convergencia prematura a una solución, situación que tiene lugar cuando los agentes dejan de investigar nuevas soluciones y se centran en recorrer las soluciones ya descubiertas.

En este comportamiento mostrado por el hormiguero tiene lugar una comunicación indirecta entre las hormigas, ya que cada una no posee información sobre lo que las demás están haciendo en un momento dado. Esta comunicación se lleva a cabo mediante los rastros de feromona; y la misma posee por un lado una naturaleza física dada por una modificación del entorno físico visitado por los insectos y por otro una naturaleza local ya que solamente puede ser accedida por los insectos que visitan el lugar donde se encuentra (o lugares cercanos a éste).

Al utilizar agentes artificiales este concepto es fácilmente realizable, por lo general haciendo uso de "variables de feromona" asociadas a los estados del problema que son visitados por los agentes, estos estados a su vez son asociados a variables de estado y los agentes tienen solamente acceso local a dichas variables. Cada agente construye una solución o componente de la misma, comenzando a partir de un estado inicial seleccionado de acuerdo a ciertos criterios dependientes del problema. Al construir su solución, cada agente recoge información tanto de las características del problema como de su propia actividad. La información brindada por cada agente se utiliza, posteriormente, para modificar el modo en que el problema total será visto por la totalidad de los agentes.

Debe tenerse presente que los agentes artificiales tienen una doble naturaleza, ya que por un lado son abstracciones de las hormigas reales y por otro han sido enriquecidos con ciertas capacidades para las que no se tiene una contraparte natural. Esto último para hacerlos más efectivos y eficientes en la búsqueda de la solución que se pretende encontrar.

Ciertas suposiciones que suelen hacerse son:

- Que el monto de feromona es proporcional al número de hormigas que han pasado por el lugar donde ésta se encuentra depositada.
- Que no existe evaporación de la feromona liberada por las hormigas (cosa que si sucede en el mundo de las hormigas reales).

Ant Systems ha proporcionado aportes en lo que se refiere al marco teórico dentro del que luego se diseñan los algoritmos particulares. A partir del análisis de sus características básicas se ha definido inclusive una metaheurística más general, denominada ACO (Ant Colony Optimization) de la que Ant System puede verse como un caso particular. En ella se utiliza una colonia de agentes artificiales relativamente simples para encontrar buenas soluciones a difíciles problemas de optimización combinatoria, asignando entre éstos los recursos computacionales.

Dentro de la metaheurística ACO se encuentran ciertas familias de algoritmos, las siguientes son algunas de ellas:

- Ant System. El primer algoritmo ACO, desarrollado por Dorigo y colegas en 1991 aplicado en sus orígenes al problema TSP (Travelling Salesman Problem). Esto hace que Ant Systems sea la primer familia en desarrollarse.

En caso de que se desee profundizar en trabajos de Dorigo remitirse al siguiente material en italiano: Dorigo M.: Ph.D. thesis "Optimization, Learning and Natural Algorithms", Dipartimento di Elettronica, Politecnico di Milano, Italia 1992.

En cambio, si lo que se desea es examinar trabajos de Dorigo y colegas sugerimos Dorigo M., Maniezzo V., and Colorni A., "Positive feedback as a search strategy.", Technical Report 91-016,Dipartimento di Elettronica,Politecnico di Milano, Italia 1991.

- Ant Colony System. Desarrollada en 1996 con el objetivo de mejorar la eficiencia mostrada por los algoritmos pertenecientes a la familia Ant System.

La primer referencia que sugerimos por este tema es Gambardella, L.M. and Dorigo, M.,"Solving symmetric and asymmetric TSPs by Ant colonies.", pp. 622 –627, Proceedings of the IEEE Conference on Evolutionary Computation ICEC96 ,.IEEE Press 1996.

Como otra referencia relacionada puntualizamos a Dorigo M. and Gambardella L. M., "Ant colonies for the traveling salesman problem.", pp. 43:73 –81, BioSystems, 1997.

Finalmente señalamos a Dorigo M. and Gambardella L.M., "Ant colony system: A cooperative learning approach to the traveling salesman problem.", pp. 1(1):53 –66, IEEE Transactions on Evolutionary Computation, 1997.

- Max-Min Ant Systems. Desarrollada en 1997. Su filosofía base es básicamente igual a la de la familia Ant System pero en ella los rastros de feromona no son actualizados directamente por los agentes; estando además, restringidos a un intervalo de valores entre un mínimo y un máximo, teniendo como valor inicial el máximo de este intervalo.

Como primer reseña por este tema: Stützle, T. and Hoos, H.,"The MAX –MIN Ant system and local search for the traveling salesman problem", pp. 309 –314, Proceedings of IEEE-ICEC-EPS '97 IEEE International Conference on Evolutionary Computation and Evolutionary Programming Conference ,IEEE Press 1997.

Como última referencia relacionada con Min-Max Ant Systems puntualizamos a Stützle, T. and Hoos, H.,"Improvements on the ant system:Introducing MAX –MIN ant system.",pp 245 –249, Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms, Springer Verlag, Wien 1997.

Como puntualización de algunas de las aplicaciones que se han dado a Ant Systems es posible mencionar: problemas relacionados al ruteo de vehículos, coloración de grafos y al ruteo en redes de comunicación.

4.1.2. Algoritmos Genéticos

La fuente de inspiración que proporcionó la base para el desarrollo de esta metaheurística fue la evolución que muestran las especies vivas, evolución que provoca la adaptación de los individuos de éstas, a las necesidades que les impone el medio en el que se desenvuelven. El cumplimiento de estas necesidades implica una fuerte competencia y un alto grado de especialización por parte de los individuos; donde los más aptos son quienes tienen la oportunidad de desarrollarse y reproducirse transmitiendo así características propias a los integrantes de la nueva generación, características que

eventualmente son las responsables del éxito de los progenitores en la lucha por la sobrevivencia.

Esta metaheurística fue introducida por vez primera por Holand en 1975 como una filosofía para diseñar algoritmos de búsqueda altamente robustos, y en la misma, la resolución de un determinando problema de optimización se traduce en la búsqueda del individuo más apto presente a lo largo de la evolución de una determinada población de individuos. En este caso cada individuo representa de alguna manera una propuesta de solución al problema objetivo.

Usualmente en la mayoría de los algoritmos genéticos existen dos componentes principales que son dependientes del problema en cuestión: la codificación del problema y la función de evaluación.

Al aplicar un algoritmo basado en esta filosofía el primer paso consta en generar una población inicial. Usualmente cada miembro de esta población será codificado como una cadena de caracteres binaria de largo L que corresponde a la codificación de una solución propuesta para el problema. Por lo general suele hacerse referencia a esta cadena utilizando el término genotipo (Holand 1975), o alternativamente "cromosoma" (Schaffer 1987). En la mayoría de los casos esta población inicial es generada al azar.

Si se desea examinar el trabajo de Holland 1975 remitirse a Holland, J.H.:"Adaptation in Natural and Artificial Systems", The University of Michigan Press, 1975.

Si se desea examinar el trabajo de Schaffer 1987 remitirse a Schaffer, J.D.:"Some Effects of Selection Procedures on Hyperplane Sampling by Genetic Algorithms", Genetic Algorithms and Simulated Annealing, L. Davis ed. Pitman 1987.

Luego de que se dispone de una población inicial cada individuo de ésta (codificado como cadena de caracteres) es entonces evaluado, y le es asignado un valor de aptitud, comúnmente llamado de "fitness" en la literatura, que es un indicador de la primacía que este tiene para reproducirse.

Algunas veces las nociones de “evaluación” y “aptitud" son utilizadas indistintamente, sin embargo es útil distinguir la función de evaluación y la función de aptitud usadas por un algoritmo genético. La función de evaluación o función objetivo provee una medida de la calidad del individuo con respecto a un conjunto particular de restricciones, mientras que la función de aptitud transforma esa medida de calidad en oportunidades de reproducción para el individuo en cuestión. Esto es, mientras que la aptitud de un individuo codificado como una cadena de caracteres es siempre definido con respecto a otros miembros de la población actual, la evaluación es independiente de la evaluación de los demás individuos.

Es útil ver la ejecución de un algoritmo genético como un proceso de dos etapas. Como punto de partida se tiene la población actual; sobre la que posteriormente se aplica una selección de individuos con el objetivo de crear una población intermedia. Sobre esta última se aplican procedimientos de recombinación y de mutación de individuos, obteniéndose una nueva población que constituirá la población siguiente obtenida a partir de la población actual. Este proceso que comienza a partir de la población actual y culmina al obtener la población siguiente constituye el proceso que da lugar a una nueva generación durante la ejecución de un algoritmo genético y esta dado por la aplicación de procesos de evaluación, selección, recombinación y mutación.

Una habilidad absolutamente distinguible de los algoritmos genéticos es la de enfocar su atención en la zona más promisoria del espacio de soluciones asociado al problema que se pretende resolver. Ésta se debe a la facultad de combinar individuos que representan soluciones parciales para el problema haciendo prevalecer los más atractivos, habilidad inherente a los algoritmos genéticos en general. En primer lugar cada individuo de la población es evaluado con el objetivo de determinar una medida del aporte proporcionado por él (es decir, su aptitud); en segundo lugar los individuos a combinar son ordenados de acuerdo a esta medida de aporte y este orden es utilizado al seleccionar que individuos se reproducirán con que otros.

El mecanismo clásico de reproducción de individuos procede de la siguiente manera; en primer lugar las cadenas de caracteres a reproducir se alinean una al lado de la otra; posteriormente de forma aleatoria se selecciona un punto de corte, punto por el que se fraccionará cada cadena; y finalmente las porciones de las cadenas que se encuentran a la izquierda de dicho punto de corte son intercambiadas dando de esta manera lugar a dos nuevos individuos, individuos que serán los descendientes de los que acaban de reproducirse.

Por otra parte, generalmente durante la mutación de un individuo se procede a alterar, básicamente al azar, alguna de los componentes binarios de la cadena que ésta codifica. De esta manera se obtiene un nuevo individuo representado por esta cadena modificada. Lo destacable de la mutación es que ofrece la posibilidad de descubrir nuevas regiones promisorias en el espacio de soluciones introduciendo una mayor diversidad en la población de individuos considerados.

4.1.3. GRASP

Esta metaheurística fue presentada en primer instancia por T. A. Feo y M. G. C. Resende en 1989, en su trabajo:

T. A. Feo; M.G.C. Resende:"A probabilistic heuristic for a computationally difficult set covering problem Operations Research Letters", pp. 8:67–71, 1989.

Posteriormente, los mismos autores generaron el primer tutorial sobre esta metaheurística:

T. A. Feo, M.G.C. Resende: "Greedy randomized adaptive search procedures.", Journal of Global Optimization, pp. 6:109–133, 1995.

El nombre que se ha dado a esta metaheurística, GRASP, se debe a las siglas en inglés para identificar un proceso de búsqueda adaptativo, aleatorizado y goloso (Greedy Randomize Adaptative Search Procedure).

En esta metaheurística están presentes algunas de las ventajas de los algoritmos greedy puros y también de procedimientos de construcción aleatoria.

Este es un proceso iterativo en el que cada iteración consta básicamente de una fase de construcción y una fase de búsqueda local. En la fase de construcción se utiliza un procedimiento greedy aleatorizado para construir una solución factible para el problema en cuestión. La fase de búsqueda local trabaja luego a partir de esta solución, y de forma iterativa intenta mejorarla sucesivamente; finalizando al llegar al punto en el que no es posible encontrar una mejor solución. Posteriormente, antes de finalizar el ciclo, esta mejor solución devuelta por la búsqueda local es comparada con la mejor solución encontrada hasta el momento almacenándose la mejor de ambas soluciones. De esta manera culmina un ciclo GRASP, luego del cual es posible repetir un nuevo ciclo en caso de que el criterio de parada establecido aún no se haya alcanzado.

A continuación se muestra un pseudocódigo que ejemplifica la estructura general típica de un algoritmo implementado sobre la base de la metaheurística GRASP:

```
PROCEDURE GRASP();
        ENTRADA_PROBLEMA();
                // Codifica la especificación del problema en términos de las
                // estructuras internas utilizadas por el algoritmo.

        MIENTRAS NoSeCumplaCriterioDeParada()  HACER
                ConstruirSolución(S);
                S*=BusquedaLocalGrasp(S);
                ActualizarSolucion(Sopt, S*);
                        // En caso de que la nueva solución encontrada sea mejor
                        //  que la mejor solución encontrada hasta el momento esta
                        // última es actualizada para almacenar de esta manera la
                        // mejor solución.
        FIN MIENTRAS
        DEVOLVER(Sopt);
FIN PROCEDURE GRASP
```

El procedimiento que implementa la fase de construcción construye cada solución factible un elemento por vez. En cada iteración de la construcción el próximo elemento a agregar se determina ordenando el conjunto de todos los elementos factibles de ser elegidos, y luego eligiendo uno de ellos al azar; donde por lo general el orden de estas componentes determina que para ciertos elementos la chance de ser elegidos sea mayor a la de otros. Este ordenamiento suele materializarse en una lista de candidatos ordenada con respecto a una cierta función golosa (greedy) que estima el beneficio de seleccionar cada uno de los elementos.

Esta elección de componentes constituye la componente probabilística de un procedimiento GRASP, y se caracteriza por la elección aleatoria de uno de los mejores candidatos en la lista, pero no necesariamente el mejor. Este hecho ocasiona que con cada ciclo se obtengan soluciones en diversas áreas del espacio de soluciones y contribuye a la diversificación de la metaheurística en su búsqueda de la solución óptima.

La solución obtenida luego de la fase de construcción no necesariamente es la solución óptima al problema, seguramente ni siquiera será una solución óptima local en el espacio de soluciones; por ello se aplica a la misma un procedimiento de mejora que es implementado por la componente de búsqueda local de GRASP.

Esta componente de búsqueda local trabaja en forma iterativa tras reemplazar sucesivamente la solución actual por una mejor solución perteneciente a su vecindad en el espacio de soluciones. Este proceso finaliza al alcanzar una solución para la que no es posible encontrar una mejor solución vecina, es decir, al alcanzar una solución que es un óptimo local. Un aspecto importante al diseñar el procedimiento de búsqueda local es el elegir adecuadamente la noción de vecindad y el intentar utilizar técnicas eficientes de búsqueda en la misma.

Es de notar aquí que la calidad de la solución inicial es importante para obtener buenas soluciones finalizada la búsqueda local, por esta razón es deseable que la fase de construcción produzca buenas soluciones iniciales para la búsqueda para que de esta manera aumente la efectividad del algoritmo implementado.

Usualmente se siguen ciertas consideraciones generales al diseñar el constructor de un algoritmo GRASP:

- Disminuir la complejidad al diseñar e implementar el constructor, para de esta manera permitir un mayor número de iteraciones al algoritmo.
- Dedicar un buen esfuerzo en el diseño de la mecánica que se utilizará para construir la solución factible, en particular en el mecanismo que permite al constructor guiarse de alguna manera con datos referentes a las soluciones encontradas en iteraciones anteriores para mejorar la calidad de la solución que ha de construir.

- Utilizar estructuras de datos eficientes e implementaciones que presten atención a la performance; teniendo en cuenta que, en el tiempo consumido por una búsqueda local a partir de una solución inicial puramente aleatoria, es común poder construir varias soluciones iniciales con aleatoriedad controlada y luego aplicarles las búsquedas locales correspondientes.

Los aspectos más distinguibles, entonces, de la fase de construcción son los siguientes:

- La elección del próximo elemento a ser colocado en la solución es determinada ordenándose todos los elementos disponibles en una lista de candidatos de acuerdo a una función golosa. Esta función golosa mide el beneficio miope local de escoger el candidato.
- La heurística es adaptativa pues los beneficios asociados a cada elemento son actualizados (recalculados) de forma de tener en consideración la elección hecha en la iteración anterior, o en iteraciones anteriores.
- La heurística es probabilística pues en la fase de construcción escoge un candidato al azar entre los mejores de la lista de candidatos a integrarse a la solución que se está construyendo. Esto permite que se generen soluciones diferentes en cada iteración GRASP, sin necesariamente comprometer el poderío de la componente golosa adaptativa de la heurística. Suele utilizarse el término RCL (Restricted Candidate List) para referirse a esta lista de candidatos.

En lo referente a la fase de búsqueda local se puede destacar lo siguiente:

- La fase de construcción no necesariamente devuelve una solución que sea óptimo local aún con relación a las vecindades más simples.
- Casi siempre resulta beneficioso aplicar una búsqueda local en la vecindad de la solución construida a los efectos de mejorarla.
- La búsqueda local no tiene porque ser exhaustiva.

¿Que hace buena a una búsqueda local?

- Escoger apropiadamente la vecindad.
- Disponer de técnicas eficientes de búsqueda en la vecindad, es decir, técnicas rápidas y con buena calidad en la selección.
- Que la fase de construcción provea soluciones con buena calidad. Notar que el esfuerzo insumido en la búsqueda local puede ser exponencial, pero que, empíricamente la eficiencia de esta mejora con la calidad de la solución inicial.

En lo que refiere a GRASP en general:

- Es difícil llevar a cabo un análisis formal, básicamente dada la aleatoriedad involucrada en su funcionamiento.
- Se puede ver a GRASP como una técnica de muestreo repetitivo. Donde cada iteración GRASP, produce una solución que es una muestra de una distribución (desconocida) de todas las soluciones posibles.

- La esperanza y la varianza de esta distribución está en función de la RCL considerada.
  - Si el número de elemento de la lista de RCL es igual 1 (|RCL|=1): Todas las soluciones son iguales teniendo esperanza alta y varianza cero.
  - Si el número de elemento de la lista de RCL aumenta se tiene: soluciones diferentes; soluciones que resultan no tan buenas respecto a cuando la |RCL|=1; la esperanza es menor a cuando |RCL|=1; la varianza aumenta; la mejor solución es mejor que la media.

Facilidades de implementación

Ciertas características facilitan la implementación de GRASP:

- Se tiene un reducido número de parámetros para inicializar y armonizar.
  - Restricción del |RCL| llevada a cabo con un parámetro que acota su tamaño,
  - Escasos parámetros necesarios para controlar el grado de golosidad y aleatoriedad en el algoritmo de construcción.
  - Número de iteraciones GRASP.

  Por esta razón no se insume demasiado tiempo en el ajuste de parámetros.
- Existen funciones golosas para casi todos los problemas de optimización combinatoria, lo cual facilita el diseño e implementación del constructor GRASP.
- En caso de que se tengan bien definidas la noción de vecindad y de búsqueda en la misma el diseño e implementación de la búsqueda local se hace de una forma casi inmediata.
- Obtener versiones paralelas de algoritmos GRASP es una tarea relativamente simple. Algunas de las alternativas más simples para diseñar e implementar una versión paralela son:
  - Distribución de problemas: dividir el procesamiento en dos procesadores, uno dedicado a la construcción y otro a la búsqueda local.
  - Distribución de iteraciones: dividir el número total de iteraciones GRASP entre varios procesadores.

4.1.4. Neural Networks

El desarrollo de teorías modernas sobre el aprendizaje y el procesamiento neuronal, así como la aparición de las primeras computadoras digitales se dieron alrededor del mismo momento, a fines de la década de 1940. Entonces surgió en forma natural la idea de modelar, utilizando computadoras, el comportamiento de neuronas y grupos de neuronas, estos últimos denominados entonces como redes neuronales (Neural Networks por su nombre en inglés), lo que posteriormente dio lugar a los primeros sistemas neuronales artificiales (ANS de Artificial Neural Systems, por su sigla en inglés).

Estos sistemas neuronales artificiales surgieron a fines de los 1950s, y fueron inspirados por el conocimiento disponible en su momento sobre el

funcionamiento del cerebro humano. Se atribuye a Frank Rosenblatt los primeros de estos trabajos, aplicado a un dispositivo denominado PERCEPTRON.

Se ha reportado que las redes neuronales constituyen una herramienta poderosa para la resolución de problemas de predicción, clasificación y reconocimiento de patrones. Sin embargo no han sido tan exitosas cuando son aplicadas a problemas de optimización y no son competitivas con las mejores metaheurísticas al aplicarse a problemas de optimización combinatoria.

Una red neuronal podría esquemáticamente definirse como una colección de procesadores paralelos conectados en la forma de un grafo dirigido (en el que a cada procesador le corresponde un nodo del grafo y donde una arista indica que la salida de uno de los procesadores sirve como entrada a otro), organizado de una manera tal que la estructura de la red resultante y la manera en que cada procesador procesa las señales de los que se comunican con él, permite la resolución del problema.

Clásicamente las redes neuronales se estructuraban en capas, es decir, se dividía el conjunto de procesadores (de ahora en más nos referiremos a los mismos utilizando el término neurona) en subconjuntos; cada uno de los cuales se denominaba capa. Una red neuronal de este tipo, se estructura de la siguiente manera:

- Las neuronas que conforman cada capa no se interconectan entre sí.
- La construcción se lleva a cabo de la siguiente manera:
  i. Se selecciona una de las capas, denominada capa de entrada, y se interconecta la misma con una sola capa de neuronas haciendo que cada neurona de esta capa vuelque su salida a cada neurona de la capa seleccionada como capa siguiente.
  ii. Luego, utilizando la misma filosofía para interconectar las capas, se van conectando en cadena las demás capas de modo que para cada neurona de cada nueva capa todas las neuronas pertenecientes a la capa anterior vuelquen su salida sobre la misma.
  iii. La última capa, denominada capa de salida, se interconecta a las demás utilizando la misma mecánica de conexión que la mencionada en los puntos anteriores, pero su salida no es utilizada como entrada para ninguna capa sino que se interpreta como una codificación de la solución provista por la red neuronal para la instancia del problema particular al que se ha aplicado.

- Cada neurona asocia un peso a cada entrada recibida por cada neurona de la capa anterior, el cual se puede interpretar como una medida de su sensibilidad a los estímulos provenientes de cada neurona que le provee información para procesar. Esta sensibilidad puede variar dependiendo de la credibilidad de la neurona receptora con respecto a la neurona emisora. Entonces, luego de que una neurona recibe los estímulos de todas sus neuronas de entrada, lleva a cabo una suerte de suma ponderada de dichos estímulos utilizando los pesos asociados a cada neurona de entrada como los pesos involucrados en dicha suma, y en caso de que el resultado de esta suma supere cierto umbral propagará una señal a todas las neuronas de la capa siguiente.

La filosofía clásica de trabajo con las redes neuronales es la siguiente:

- En primer lugar se tiene una fase de entrenamiento en la que sistemáticamente se prueba la red con diversas instancias del problema que se pretende resolver. Con cada prueba se compara el resultado provisto por la red neuronal con el resultado exacto (o el mejor resultado conocido) para la instancia en cuestión, y se procede a un ajuste de los pesos que cada neurona asocia a sus neuronas de entrada, de modo que al aplicar nuevamente esa misma instancia del problema, el resultado devuelto por la red este más cercano al resultado deseado. Este ajuste de pesos es lo que implementa el aprendizaje de la red neuronal y se lleva a cabo de una manera ordenada, utilizando un procedimiento matemáticamente justificado identificado con el término "Back Propagation".

Por un mayor detalle en lo que refiere a "Back Propagation" remitirse a Freeman, James A. and Skapura, David M. "Neural Networks: Algorithms, Applications, and Programming Techniques", pp.89-126, Addison Wesley, July 1992.

Llevar a cabo el entrenamiento de una manera adecuada dista de ser una tarea sencilla. En primer lugar no existe un método que permita seleccionar los casos de prueba con los que se entrena la red neuronal. Por otro lado las redes neuronales tienen un cierto límite en su capacidad de aprendizaje ya que se ha visto empíricamente que al entrenar la red con demasiados casos su desenvolvimiento, en términos de la calidad de las soluciones encontradas, se ve afectado. Por último, de todos los "problemas tipo" a los que la red neuronal se ve enfrentada esta tiende a ajustarse a los últimos casos en detrimento de los primeros; esto ocasiona que la red tienda a "olvidar" los primeros "problemas tipo" presentados, imponiéndose así un límite natural en lo que a tipos de problemas para entrenamiento se refiere. Todos estos factores hacen del entrenamiento una actividad en la que prima más la intuición que el método en lo que refiere a su planificación.

- Luego la red neuronal se pone a disposición, es decir, cuando se considera que el desempeño de la red es suficientemente bueno se utiliza la misma, frente a instancias del problema para las que se desconoce su solución exacta o aproximada.

Si bien lo antes expuesto ejemplifica la estructura clásica de las redes neuronales, existen numerosas variantes que básicamente difieren entre si en la forma en que se interconectan las neuronas y en la forma en la que cada neurona procesa los estímulos que recibe para luego generar el suyo propio. Siendo estas arquitecturas, como por ejemplo las redes neuronales auto-organizativas (Self-Organizing Neural Networks) más adecuadas en la resolución de ciertos problemas puntuales.

Una fuente de material para remitirse a trabajos relativos a este tipo de redes neuronales la constituyen los trabajos de T. KOHONEN, en particular: KOHONEN, T.: The Self-Organizing Map", pp. 1464–1480, Proceedings of the IEEE 78, 1990.

La idea de utilizar redes neuronales para resolver problemas de optimización combinatoria se originó en 1985, cuando Hopfield y Tank demostraron que el problema TSP- Travelling Salesman Problem puede ser resuelto utilizando una Neural Network tipo Hopfield. Este trabajo, aunque muy controvertido en su momento, sirvió como puntapié inicial para la utilización de las Neural Networks como herramientas para atacar este tipo de problemas.

Para remitirse a este trabajo, de particular importancia en el campo de las redes neuronales, es posible referirse a HOPFIELD, J. J. and TANK, D. W: " 'Neural' Computation of Decisions in Optimization Problems", pp. 141–152, Biological Cybernetics 52, 1985.

A

4.1.5. Simulated Annealing

Fue popularizada como una metaheurística de optimización por Kirkpatrick, Gelatt y Vecchi en 1983.

Por un mayor detalle sobre este trabajo remitirse a Kirkpatrick, S., C. D. Gelatt Jr., y M.P. Vecchi: "Optimization by Simulated Annealing", pp. 671 – 680, Science V. 220, No. 4598, 1983.

Es una técnica muy general que, mientras lleva a cabo una búsqueda utilizando "movidas" probabilísticas en el espacio de soluciones asociados al problema, intenta evitar el quedar atrapada en los mínimos locales que pueda llegar a encontrar en su camino de búsqueda.

Esta es una metaheurística inspirada en el proceso de cristalización de los metales, donde:

- A una temperatura suficientemente alta, la energía del conjunto de átomos hace que el metal pase al estado líquido, cumpliéndose que al decrecer la temperatura la energía del sistema evoluciona dependiendo de la velocidad con la que se lleve a cabo el enfriamiento.

- Si el enfriamiento es muy brusco el sistema alcanza un estado de congelamiento manteniendo una alta energía interna. Esa situación se denomina caos y se corresponde con el óptimo local de los problemas de optimización.
- Si el enfriamiento se hace lentamente los átomos disminuyen progresivamente su velocidad, encontrando para cada temperatura un estado de equilibrio; de esta manera para cada temperatura alcanzada la energía interna es mínima. Así cuando los átomos alcanzan el estado de congelamiento se obtiene un cristal con la mínima energía interna.

Teniendo en cuenta estas propiedades sobre el enfriamiento de los metales, se ha desarrollado un método para obtener metales sólidos con una estructura cristalina interna perfecta o casi perfecta. Básicamente puede describirse de la siguiente manera:

- En primer lugar el metal es calentado hasta que la temperatura sea la suficiente para fundirlo, dejándolo de esta manera en estado líquido.
- Posteriormente es enfriado lentamente. Haciéndolo a la velocidad adecuada, los átomos tendrán una oportunidad mayor de reordenarse en una estructura cristalina cercana a la perfecta.

En la metaheurística Simulated Annealing se mantiene una cierta analogía con este método utilizado para obtener metales perfectos; dentro de esta similitud el obtener una solución de costo mínimo para el problema a resolver es análogo a obtener una estructura cristalina perfecta para el metal en proceso.

Continuando con esta analogía, la función de costo asociada al problema (es decir la función que se pretende minimizar), función que mide el costo de cada solución posible, es vista como la función que mide la energía del sistema en el estado representado por la solución a la que se aplica; de esta manera un estado del sistema con una estructura cristalina perfecta representaría una solución al problema objetivo que minimiza la función de costo asociada a dicho problema, y de esta manera se correspondería con la solución buscada para el problema en cuestión.

Los algoritmos implementados a partir de la filosofía promulgada por esta metaheurística, pertenecen a la categoría de algoritmos que, en inglés, se han dado en llamar “probabilistic hill-climbing algorithms”. Es decir, pertenecen al conjunto de algoritmos que en su recorrido por el espacio de soluciones del problema, en búsqueda de la solución de costo mínimo, se permiten con cierta probabilidad la libertad de transitar por estados de dicho espacio de soluciones que en lugar de mejorar empeoran el mejor costo obtenido hasta el momento. Esta característica es importante pues ocasiona que los algoritmos implementados tiendan a evitar el quedar atrapados en el entorno de algún óptimo local mientras inspeccionan el espacio de soluciones.

Al utilizar esta metaheurística para atacar un problema se hace necesario tomar ciertas decisiones de implementación. Por lo general deberá decidirse:

- Cual será el espacio de estados.
- Cual será la estructura de vecindario a utilizar.

- Cual será la función de costo a utilizar.
- Cual será el estado inicial a partir del cual se lleva a cabo el recorrido en el espacio de estados.
- Cual será la estrategia de enfriamiento, o sea:
  - Cual será la temperatura inicial.
  - Cual será el plan de enfriamiento, esto es, como se reducirá la temperatura
  - Cual será la condición de equilibrio, esto es, que condición se utilizará para determinar que el sistema se encuentra en estado de equilibrio a la temperatura actual.
  - Cual será el punto de congelamiento, esto es, cual será la temperatura que al alcanzarse determine que se da por finalizada la búsqueda en el espacio de estados.

No existen consideraciones, datos o sugerencias que asistan en esta toma de decisión; por lo que el definir cual será la estrategia de enfriamiento se convierte en un arte más que en una ciencia y por lo general requiere de experimentación sucesiva al estilo "ensayo y error" para determinar los mejores parámetros.

En la literatura dedicada a esta metaheurística se reporta que en la práctica Simulated Annealing tiene un buen desempeño si se invierten grandes esfuerzos computacionales en la ejecución de los algoritmos; y se sugiere que, en general, sea utilizado como un último recurso para atacar un problema de optimización, recomendándose su uso para aquellos casos en los que no se tiene un entendimiento suficiente del problema a resolver. De todas maneras, se reporta que ha probado ser una herramienta poderosa cuando es aplicada juiciosamente, y cuando en la elaboración de los algoritmos es tenida en cuenta toda información disponible sobre el problema objetivo. [G1995]

4.1.6. Tabu Search

Los orígenes de Tabu Search se remontan a la década de 1970, sin embargo fue presentada por primera vez en su forma actual por Glover [G1986]. A fines de la década de los '80 y comienzos de los '90 aparecieron nuevas recopilaciones de las ideas básicas y esfuerzos de formalización por parte de Hansen, Glover y Werra & Hertz; y a comienzos de los 90' Faigle & Kern, Glover y Fox formalizan ciertos aspectos teóricos de esta metaheurística.

Para remitirse al trabajo de Glover 1986 remitirse a Glover, F: "Future Paths for Integer Programming and Links to Artificial Intelligence Computers and Operations Research", pp. 533-549, 1986.

Para remitirse al trabajo de Hansen 1986 remitirse a Hansen, P. "The Steepest Ascent Mildest Descent Heuristic for Combinatorial Programming", - Presented at the Congress on Numerical Methods in Combinatorial Optimization, Capri - Italy 1986.

Para remitirse al trabajo de Glover 1989 remitirse a Glover, F. "Tabu Search, Part I", pp. 190-206, ORSA Journal on Computing 1, 1989.

Para remitirse al trabajo de Werra & Hertz 1989 remitirse a Hertz, A. y de Werra, D. "The Tabu Search Metaheuristic: how we used it", pp. 111-121, Annals of Mathematics and Artificial Intelligence 1, 1990.

Para remitirse al trabajo de Glover 1990 remitirse a Glover, F. "Tabu Search, Part II", pp. 4-32, ORSA Journa on Computing 2, 1990.

Para remitirse al trabajo de Faigle Y Kern remitirse a Faigle, U. y Kern, W. "Some Convergence Results for Probabilistic Tabu Search", pp. 32-37, ORSA Journal on Computing 4, 1992.

Para remitirse al trabajo de Glover 1992 remitirse a Glover F. "Private communication", 1992.

Para remitirse al trabajo de Fox 1993 remitirse a Fox B. L. "Integrating and accelerating Tabu search, simulated annealing and Genetic Algorithms", pp. 47-67, Annals of Operations Research 41, 1993.

Tabu Search es un método de búsqueda en el espacio de soluciones asociado al problema al que se aplica, caracterizado porque a partir de una solución dada avanza hacia una nueva solución perteneciente al vecindario de la primera. La toma de decisiones sobre hacia que solución vecina se debe saltar, es realizada de forma muy elaborada en comparación con la decisión que al respecto implementa el tradicional método de búsqueda en la vecindad, conocido como el método descendente.

El método descendente es el método más famoso de búsqueda en la vecindad de soluciones, y ha sido utilizado para encontrar una aproximación al mínimo de funciones sobre un determinado conjunto.

Método Descendente
Sea f una función definida sobre un cierto dominio D, función que toma valores numéricos; y sea un conjunto S $\subseteq$ D. El objetivo del método descendente es determinar el valor del mínimo de la función f al restringirla sobre el conjunto S.

Formalmente, el objetivo de este método es determinar el valor

$$m = \min_{x \in S} f(x)$$

Los pasos que componen este método pueden enumerarse de la siguiente manera:

1. Se elige un elemento inicial $i \in S$.
2. Siendo N(i) el vecindario de i, se determina el mejor $j \in N(i)$, es decir, el j tal que **$f(j) = \min_{x \in N(i)} f(x)$**.
3. Si f(j) >= f(i) entonces se ha llegado al fin y el valor devuelto por el método es f(i); en caso contrario se ejecuta la asignación i=j y se vuelve al paso 2.

Este método es esencialmente simple, característica destacable por las ventajas que esto conlleva al momento de desarrollar algoritmos que lo implementen, pero es incapaz de saltar los mínimos locales en su búsqueda del óptimo global ya que la búsqueda que lleva a cabo se detiene al encontrar el primer mínimo local, aunque éste no sea precisamente el mínimo global de la función objetivo en el conjunto de interés. Por esta razón, exceptuando algunos casos especiales de convexidad de la función que se pretende minimizar, el uso de procedimientos descendentes como el mencionado anteriormente es generalmente frustrante ya que luego de finalizados el valor encontrado por los mismos puede distar bastante (en términos del valor de la función f) del valor asociado al óptimo global deseado.

Si se pretende mejorar la eficacia en el proceso de exploración, al momento de tomar la decisión de que salto llevar a cabo en el espacio de soluciones se debe tener a la vista no solo la información local sino también alguna información relacionada con el proceso de exploración. La información mantenida por Tabu Search, así como la forma en que hace un uso sistemático de la misma para disponer de una guía en la elección de la próxima solución a visitar, constituyen un distintivo esencial de esta metaheurística.

Al continuar el razonamiento abierto por el análisis del método descendente resulta evidente que cualquier proceso iterativo de exploración debe, en algunos casos, aceptar un movimiento que no mejora el pasaje de la solución actual a la solución siguiente con el propósito de escapar de los mínimos locales encontrados durante la búsqueda. Simulated Annealing dispone de mecanismos para lograr este comportamiento, pero estos mecanismos no permiten guiar la elección del nuevo movimiento, Tabu Search en contraste elige en cada paso el mejor movimiento posible en función del vecindario.

Tan pronto como se permiten movimientos donde no se producen mejoras al pasar de la solución actual a la solución siguiente surge un nuevo problema, ya que se corre el riesgo de visitar una solución en más de una oportunidad, y por lo tanto la posibilidad de que el algoritmo implementado quede preso en un ciclo infinito, posibilidad que no se debe dejar de considerar.

Frente a este nuevo problema, es útil el uso de memoria para prohibir movimientos que pueden llevar a soluciones recientemente visitadas. En Tabu Search esta memoria se implementa mediante lo que se ha dado en llamar “Lista Tabu”, la cual consta de las soluciones que se han visitado recientemente.

Para evitar repetir alguna de las últimas k soluciones visitadas es necesario disponer de una memoria de tamaño k al menos, por lo que, en caso de tener una cota k para el tamaño de la Lista Tabu, situación que se da en la práctica al no disponerse de memorias de tamaño infinito, sólo se puede garantizar el evitar ciclos de tamaño a lo sumo k.

Sin embargo, el uso dado a esta "lista tabu" debe ser cuidadoso, pues en caso de aplicar estrictamente la filosofía de no repetir soluciones ya visitadas, se impone un patrón de búsqueda que excluye completamente de la misma ciertas soluciones. Las soluciones que nunca serán visitadas y a las que se cataloga como "soluciones en estado tabu", son todas aquellas soluciones vecinas a alguna que aún pertenece a la lista tabu por haber sido visitada recientemente. De esta forma se limitaría por error la intensidad de la búsqueda en el entorno de soluciones atractivas.

Una estrategia que suele utilizarse para lograr atenuar esta debilidad, se denomina "criterio de aspiración", consistente en ignorar el estado tabu de las soluciones para las que se determina que desde las mismas es posible alcanzar una solución mejor que desde la actual.

Formalmente, sea f la función objetivo y h una función de aspiración, entonces se cumple que:

- h asocia un valor v a cada valor u alcanzado por f, el que es una aproximación del mejor valor que puede obtener el algoritmo a partir de una solución s para la que f(s)=u
- siendo s la solución actual y s' la mejor solución vecina, se lleva a cabo el salto desde s hacia s', aún en el caso de que esta última este en estado tabu, si se cumple que el valor f(s') resulta favorable con respecto a h(f(s)).

Durante la búsqueda de la solución óptima suele ser oportuno intensificar la exploración en las regiones del espacio de soluciones que parecen ser promisorias. Una manera en la que esta idea puede plasmarse en Tabu Search consiste en introducir un término adicional en la función objetivo; término que penalizaría las soluciones que se encuentren distantes de la presente, intensificando así la búsqueda en un entorno de la solución actual. Sin embargo, debe tenerse en cuenta que no es conveniente hacer un uso abusivo de esta técnica, pues la búsqueda tenderá a estancarse en una sola región dentro del espacio de soluciones. Por esta razón, ésta suele aplicarse alternativamente con la función objetivo original, aplicándose la primera un número acotado de iteraciones para luego retomar con la segunda; este proceso se repite a lo largo de toda la búsqueda siempre que se considere conveniente aplicar la función objetivo afectada con el término adicional.

Una técnica complementaria a la descripta anteriormente suele utilizarse cuando se desea favorecer la exploración en nuevas regiones del espacio, potenciando de esta manera la diversificación de la búsqueda. De forma análoga a la anterior suele implementarse adicionando un término a la función objetivo; término cuya función en este caso es la de penalizar las soluciones que estén fuertemente relacionadas con la actual.

Estas dos técnicas son muy importantes y el buen uso de las mismas puede aumentar el desempeño de los algoritmos implementados. Por lo general suelen implementarse de forma combinada agregando a la función objetivo dos términos, uno como el agregado por la técnica que potencia la diversificación y otro como el de la técnica que potencia la intensificación; a cada uno de estos términos se les asigna un peso que puede ser ajustado durante el proceso de búsqueda para alternar las fases de diversificación e intensificación.

Los criterios utilizados como criterios de parada son tan importantes como los criterios utilizados para explorar el espacio de soluciones, pues establecen el momento en el que se detiene la búsqueda, de esta manera criterios muy estrictos pueden ocasionar una temprana finalización y criterios muy flexibles pueden conducir a algoritmos que nunca finalicen o que lo hagan con un excesivo tiempo de cálculo adicional al necesario. Los criterios de parada más comunes suelen ser los siguientes (o alguna combinación de los mismos):

- Que la vecindad en el próximo paso sea vacía.
- Que se haya sobrepasado el número máximo de iteraciones deseadas.
- Que el número de iteraciones desde la última mejora en la solución sea mayor a un valor dado.
- Que se disponga de evidencias de que se ha encontrado la solución óptima.

Concluyendo con esta sección es de destacar que aún se desconoce una prueba clara de la convergencia a la solución óptima en caso de utilizar Tabu Search, pero la técnica a demostrado una eficacia importante en muchos problemas prácticos.

4.2. Criterios de Selección Propuestos

En esta sección abordamos el problema de la definición de los criterios de selección que se deben utilizar al seleccionar las dos metaheurísticas más adecuadas para el problema objetivo de este taller.

El conjunto de criterios utilizado para la selección de las metaheurísticas determinó, tanto el proceso de selección en sí mismo, como las metaheurísticas que finalmente resultaron seleccionadas. Esto fue un punto clave que implicó la necesidad de definir, previamente a la definición de los criterios, las características que debe de cumplir dicho conjunto a los efectos de que la selección de las metaheurísticas sea la mejor posible.

A continuación presentamos las características que en su momento consideramos deberían de estar presentes en dicho conjunto, y posteriormente los criterios que finalmente fueron utilizados.

Hacemos especial hincapié al resaltar el hecho de que tanto en esta sección, como también en la sección siguiente, resumimos lo obtenido luego de las múltiples discusiones y decisiones que al respecto debimos llevar a cabo para hacer tangible y formalizar, en la medida de lo posible, la decisión de cuáles serían las dos metaheurísticas más adecuadas para este taller. Objetivo difícil si se quiere cuando este debe realizarse en tiempos acotados y con conocimientos básicos del tema en cuestión.

La presente sección se divide en dos subsecciones. En la sección 4.2.1 planteamos una discusión referente a las características deseadas para los criterios. Una presentación de los criterios finalmente concebidos aparece en la sección 4.2.2.

4.2.1. Características Deseadas para los Criterios

A continuación presentamos las características que consideramos deberían de tener los criterios que utilizaríamos en la selección de las dos metaheurísticas más adecuadas para el taller, para que luego de la selección resultaran elegidas las mejores metaheurísticas en este sentido.

Bien especificado
El conjunto de criterios debe estar bien especificado, es decir, cada uno de los criterios que conforma dicho conjunto deberá estar especificado sin ambigüedades, de una forma clara y entendible; y además de la manera más completa posible.

Independencia
Los criterios deben ser independientes entre sí, es decir, no puede suceder que un criterio resulte de la combinación de uno o más criterios.

Completitud
El conjunto de criterios debe ser lo más completo posible, es decir, los criterios que lo conformen deberán en conjunto tener en cuenta el mayor número de características de las metaheurísticas consideradas y cada uno de ellos deberá definir cualquier concepto o terminología nueva que introduzca.

Consistencia
Los criterios que conformen el conjunto de criterios que se aplicarán deben ser consistentes entre sí, es decir, no deben existir criterios que generen contradicciones ya sea en sí mismos o con otros criterios.

Justificación rigurosa
La conformación del conjunto de criterios deberá tener una justificación rigurosa, es decir, la presencia de cada criterio en el conjunto de criterios debe estar justificada de la manera más precisa posible.

4.2.2. Criterios a Utilizar

Los criterios utilizados pueden agruparse en tres familias de criterios. La primera de ellas es el grupo de los criterios que permiten llevar a cabo la evaluación de cada metaheurística en sí misma. Por su parte, la segunda permite determinar la aplicabilidad de cada metaheurística al problema objetivo de este taller. Y por último, la tercera permite reunir toda la información relevante a los efectos de la selección que por su naturaleza no pertenece a ninguna de las familias anteriores.

Familia 1: Evaluación de la metaheurística en sí misma.

Claridad

Este criterio pretendió medir la claridad en la conceptualización de la metaheurística. La necesidad de contar con este criterio radicó en que entendimos importante valorar aquellas metaheurísticas en las que sería inmediata la codificación del problema, y las que por sus características propias permitirían además un seguimiento y control tanto en las fases de intensificación como en las de diversificación.

Documentación disponible.

Este criterio pretendió medir la calidad y diversidad de la documentación existente y accesible por nosotros referida a la metaheurística en cuestión.

Versatilidad

Este criterio midió la facilidad con la que sería posible adaptar, mejorar y/o generar nuevos algoritmos a partir de un algoritmo ya desarrollado para la metaheurística en cuestión. Con este criterio pretendimos valorar algunas metaheurísticas que tendrían la propiedad de permitir en etapas ulteriores potenciar los algoritmos llevados a cabo realizando adaptaciones mínimas o combinándolos con implementaciones de otras metaheurísticas.

Familia 2: Aplicación al problema objetivo.

Antecedentes de aplicación al problema de cálculo de confiabilidad en redes

Dado que no se tenían antecedentes de la aplicación de metaheurísticas al problema del cálculo de Diámetro-Confiabilidad en redes, apelamos a una valoración alternativa que resultó ser la aplicabilidad al problema del cálculo de la confiabilidad en redes, siendo este último un caso particular del primero.

La formalización de que el cálculo de la confiabilidad para una red es un caso particular del cálculo de la diámetro-confiabilidad puede encontrarse en Petingi L., Rodríguez, R.: "Reliability of Networks with Delay Constraints" 2001.

Aplicabilidad y complejidad de implementación.
Con este criterio se midió tanto el grado con el que para nosotros una metaheurística era aplicable al problema de Diseño de Redes Diámetro-Confiables, como también el grado de complejidad que se requeriría al momento de desarrollar algoritmos basados en los principios de dicha metaheurística.

Familia 3: Otros

Percepción general.
Este criterio fue definido al estilo criterio de clausura con el cual se describirían puntos subjetivos y/o otros, que por su naturaleza, no estaban comprendidos en los criterios anteriores y para los cuales no tenía sentido alguno identificar un criterio nuevo.

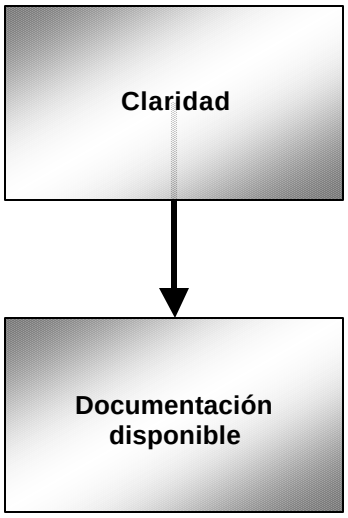
4.2.3. Importancia Relativa

Ordenamiento de criterios según prioridad

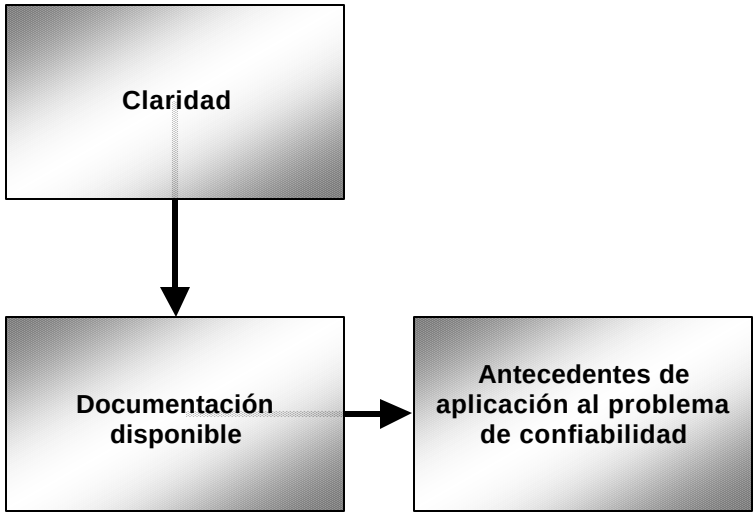
En primer lugar decidimos asignar prioridades a cada uno de los criterios aplicados sobre las metaheurísticas, asignación que de por sí ocasionó múltiples análisis. La idea fue utilizar estas prioridades para determinar luego las dos metaheurísticas más adecuadas.

Se hicieron las siguientes comparaciones entre criterios, determinándose luego el orden a partir del mismo.

Comparando el criterio “Claridad” con el criterio “Documentación disponible” podemos decir que entre los dos asignamos más prioridad al criterio “Claridad”. Esto pues en los casos de resolución de problemas, como es el nuestro, es preferible tener claros los principios fundamentales de la metaheurística a utilizar que disponer de variada documentación. Teniendo en cuenta que, esta documentación, por lo general, trata de temas locales que no aportan a la resolución de nuestro problema, precisamente por no existir material relacionado.



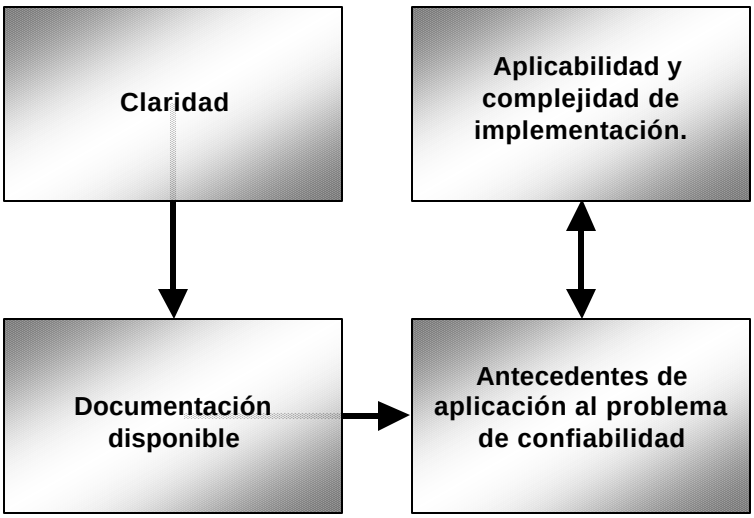
En el caso de los criterios “Documentación disponible” y “Antecedentes de aplicación al problema de confiabilidad en redes” encontramos que el primero tiene una prioridad mayor al segundo. Por un lado estamos comparando el disponer de documentación relacionada al problema objetivo, contra el disponer de antecedentes de aplicación de la metaheurística sobre un problema que si bien tiene algunas características similares al problema objetivo, es en si, esencialmente diferente. Por tanto, por más que la valoración de la metaheurística sea muy buena según el criterio “Antecedentes de aplicación al problema de confiabilidad en redes” es preferible disponer de documentación que proporcione elementos para atacar al problema real de este taller.



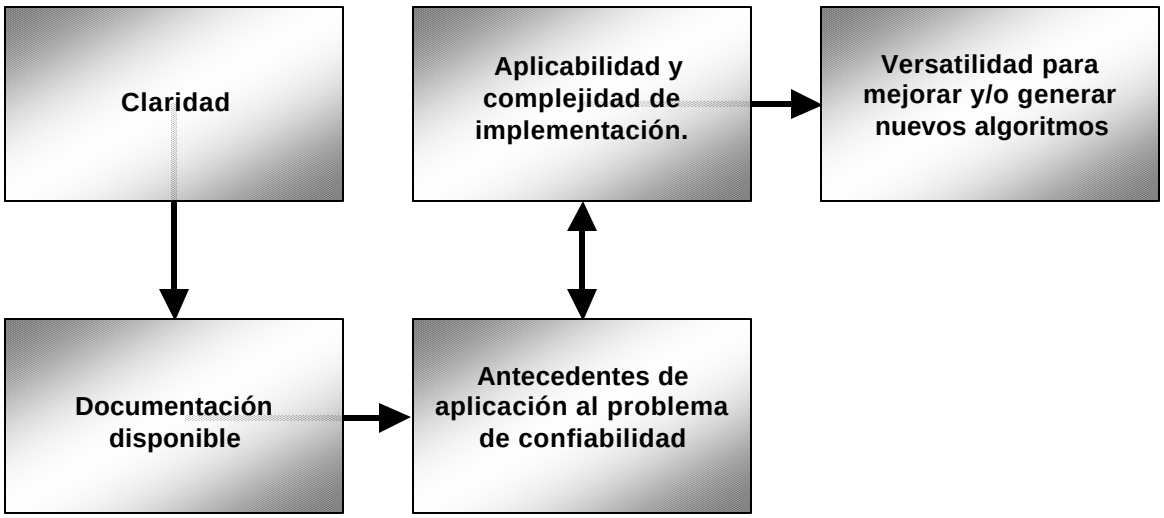
Los criterios “Antecedentes de aplicabilidad al problema de confiabilidad” y “Aplicabilidad y complejidad de implementación” resultan a nuestro entender equivalentes en prioridad al momento de clasificar las metaheurísticas, a pesar de que cada uno mide las bondades de formas diferentes.

Por un lado “Aplicabilidad y Complejidad” parece en un principio más importante, en la medida que ya se ha determinado a grandes rasgos la conveniencia o no de esa metaheurística y probablemente se tenga en mente una manera de encarar el problema con la misma. Mientras que, el criterio de “Antecedentes de aplicabilidad al problema de confiabilidad” valora la aplicación a un problema que no es exactamente el que pretendemos resolver. Notar que, por menos antecedentes que se dispongan de aplicabilidad al problema de confiabilidad, una determinación de que es probable la aplicabilidad y la facilidad de implementación serían suficientes para considerar la metaheurística.

Por otro lado, si llevamos a cabo un razonamiento opuesto, parece claro que "Antecedentes de aplicación al problema de confiabilidad en redes" debe tener una mayor prioridad que "Aplicabilidad y complejidad de implementación" en la medida que contribuye a determinar a ésta, y que resulta preferible tener datos tangibles de que la metaheurística se ha aplicado a un problema con algunas características en común al que pretendemos resolver, que el saber que bajo una cierta valoración subjetiva la metaheurística puede llegar a ser aplicable según nuestros intereses. Cabe destacar esto último aquí, pues si bien el criterio de "Aplicabilidad y complejidad de implementación" mide el grado y la complejidad con la que es aplicable cada metaheurística, esta valoración se lleva a cabo a priori, intuitivamente y a partir del análisis en abstracto de la metaheurística y del problema, lo cual constituye una valoración con alto grado de subjetividad.



Al comparar el criterio "Aplicabilidad y complejidad de implementación" con el criterio "Versatilidad para mejorar y/o generar nuevos algoritmos" notamos que la mayor prioridad debería ser asignada al primero de estos. Esta asignación en la valoración relativa entre ambos criterios se debe al hecho de que, a los efectos del presente taller es deseable el utilizar metaheurísticas que sean aplicables a nuestro problema objetivo antes que priorizar el uso de metaheurísticas versátiles en lo que respecta a modificaciones posteriores tendientes a una implementar una mejora respecto a la primer implementación. Esto es así pues poco nos aporta una metaheurística altamente versátil pero escasamente aplicable a nuestro problema objetivo; razón por la cual desde el punto de vista de los objetivos que debemos alcanzar en el presente taller, podemos ver a la "Versatilidad para mejorar y/o generar nuevos algoritmos" como una propiedad más pintoresca que útil.



De lo anterior deducimos que el orden de prioridad que debemos asignar a los criterios será el siguiente:

- Claridad
- Documentación disponible.
- Antecedentes de aplicación al problema de confiabilidad en redes, y Aplicabilidad y complejidad de implementación.
- Versatilidad para mejorar y/o generar nuevos algoritmos

A todo esto no se ha mencionado aún el criterio "Percepción general", el cual es especial en la medida que agrupa toda la información relevante a cada metaheurística, información que no pueden catalogarse dentro de los criterios anteriores. Por las características propias de este criterio, hemos considerado utilizarlo, de forma independiente, al finalizar la aplicación de los demás criterios, en el entendido que, resulte un punto importante en la validación y contraposición de los resultados obtenidos con la idea intuitiva.

4.3. Aplicación de los Criterios de Selección

4.3.1. Aplicación sobre las Metaheurísticas

A continuación analizamos las bondades y limitaciones de cada metaheurística para el problema objetivo de este taller según los criterios previamente definidos, y luego presentamos la selección de las dos más adecuadas para al problema en cuestión.

Consideraciones generales:

- A los efectos de facilitar la discusión de sí una metaheurística tiene o no características propias de paralelización, hemos introducido el término "paralelización trivial" para referirnos a ciertas variantes de paralelización (aplicables a las metaheurísticas) que por su naturaleza resultan aplicables a cualquier metaheurística. Esta "paralelización trivial" tiene, según entendemos, tres variantes:
  - Paralelizar dividiendo o reduciendo la complejidad en alguna etapa de la heurística.
  - Paralelizar teniendo instancias independientes de la misma heurística, cada una de las cuales hace una búsqueda en una determinada región del espacio de soluciones.
  - Paralelizar, si el problema lo permite, dividiendo el problema en subproblemas (los cuales serían resueltos paralelamente) y luego combinar las soluciones parciales obteniendo una solución global.

- Para ciertos criterios utilizamos una escala que mide el grado con el que la metaheurística cumple el criterio en cuestión. La escala utiliza tres valores principales: Baja, Media y Alta. La idea es utilizar una escala simple que ofrezca valores intuitivos para luego contrastar de una forma directa las diferentes metaheurísticas. Además, a dichos valores básicos hemos agregado otros, para disponer de una escala que permita expresar los matices necesarios, los que introducen una variación de los valores básicos.
  Para mantener en lo posible lo intuitivo de esta escala al incluir los nuevos valores utilizamos los símbolos de suma (+) y diferencia (-) conjuntamente con los valores básicos mencionados; donde "Valor(+)" representa un valor superior a "Valor" pero cercano al mismo e inferior al siguiente valor en la escala. En forma análoga utilizamos el símbolo "Valor(-)", pero en este caso para representar valores por debajo del valor "Valor".

Metaheurística: Algoritmos Genéticos

Claridad

Filosofía de trabajo "natural".
Criterio de búsqueda local (intensificación) claro y ajustable, diversificación clara y ajustable. La búsqueda local se puede encapsular dentro del criterio de reproducción.
Dado un problema del cual se tiene identificada la mecánica para su resolución, entendemos que existe una relación directa entre las variaciones en la mecánica de resolución y el impacto que éstas producen en la solución al problema.

Documentación disponible

Diversidad: Alta (+)
Calidad: Alta

Antecedentes de aplicación al problema de confiabilidad
Diversidad: Media
Calidad: Media

Aplicabilidad y complejidad de implementación
Aplicabilidad: Alta (+)
Complejidad: Alta (+)

Versatilidad para mejorar y/o generar nuevos algoritmos
Entendemos que Algoritmos Genéticos es la metaheurística más versátil pues permite:

- Que las soluciones iniciales sean generadas por otras metaheurísticas.
- Que durante la actividad del algoritmo sea posible enriquecerlo con soluciones encontradas por otra u otras metaheurísticas contribuyendo así a la búsqueda de la mejor solución.
- Modificar el criterio de reproducción para implementarlo utilizando otras metaheurísticas.
- La posibilidad de tener disponibles más de un criterio de reproducción los cuales pueden ser algoritmos definidos sobre la base de diferentes metaheurísticas.

Puntualización

Entendemos que el concepto de reproducción citado en la literatura, el cual hace referencia únicamente al concepto de reproducción de dos individuos, puede generalizarse a combinar uno o más de dos individuos. De forma apresurada la reproducción de un solo individuo (reproducción asexuada) puede verse como una mutación (lo cual haría desestimar la propuesta de una reproducción de un solo individuo). No obstante ello, entendemos deben verse ambos conceptos como distintos. La reproducción asexuada tendría como principio fundamental la ***creación de un nuevo ser*** *de características iguales o similares a su progenitor con el objetivo de* ***perpetuar*** *de cierta forma intacta su especie. Mientras que, con la mutación se obtiene una* ***modificación de un ser ya existente*** *cuyo objetivo, contrario al anterior, es* ***diversificar la población.***

- Dado que la población al final (gracias al criterio de selección) esta formada por soluciones de por lo menos buena calidad, permite que estas puedan ser utilizadas como entradas para otras metaheurísticas.
- Además, las características propias de Algoritmos Genéticos permiten de manera fácil, y lo que es más importante eficaz, la implementación de algoritmos paralelos en los que, la interacción entre las partes paralelas contribuye significativamente al hallazgo de la solución óptima.

Percepción general
Algoritmos Genéticos se adapta muy bien a las necesidades de nuestro taller.

Metaheurística: Tabu Search

Claridad

Fundamentos de la metaheurística claros. Filosofía natural.
Existe la flexibilidad de dejar una clara relación entre un problema y la mecánica de solución.
En lo que respecta a las facilidades que brinda de intensificación corresponde decir que se puede controlar y ajustar la búsqueda local. Mientras que, en lo que refiere a la diversificación, solo se lleva a cabo una diversificación en un entorno de la búsqueda, sin hacer saltos en el espacio de soluciones.

Documentación disponible

Diversidad: Media (-)
Calidad: Media

Antecedentes de aplicación al problema de confiabilidad

Diversidad: Baja
Calidad: Baja

Aplicabilidad y complejidad de implementación

Aplicabilidad: Media (+)
Complejidad: Media (+)

Versatilidad para mejorar y/o generar nuevos algoritmos

Podemos decir que Tabu Search no permite demasiada versatilidad, a no ser por el caso trivial que es recibir una solución dada por otra metaheurística para utilizarla como punto de partida.

Los algoritmos paralelos para Tabu Search se dividen en dos: uno que consta en comenzar a trabajar partiendo de una solución inicial y en el que la búsqueda en la vecindad es dividida entre varios procesadores, permitiendo así una reducción de la complejidad en proporción al número de procesadores. Y un segundo, consistente en implementar búsquedas independientes, cada una de las cuales comienza con una solución inicial diferente. En ambos casos entendemos que la implementación paralela no es más que la "paralelización trivial" comentada en *"Consideraciones generales"* al comienzo de esta sección.

Percepción general

La metaheurística Tabu Search es básicamente una búsqueda local, entendemos que esta más asociada con la intensificación que con la diversificación. Constituye por tanto una búsqueda local potente más que exploratoria.

Metaheurística: Simulated Annealing

Claridad

Filosofía poco natural ya que hay que asociar el problema a las temperaturas que se obtendrán a lo largo de la ejecución perdiéndose el control de lo que sucede.
No existe una clara relación entre un problema y la mecánica de solución.
Búsqueda local miope: No se tiene control de cómo se lleva a cabo la búsqueda local en el espacio de soluciones.
Diversificación miope: No se identifican claramente las regiones que se exploran y el ajuste de la exploración se hace a ciegas.

Documentación disponible

Diversidad: Baja
Calidad: Media

Antecedentes de aplicación al problema de confiabilidad

Diversidad: Baja
Calidad: Baja

Aplicabilidad y complejidad de implementación

Complejidad: Alta (+)
Nota: Es compleja la fijación de parámetros, no la implementación.
Aplicabilidad: Baja

Versatilidad para mejorar y/o generar nuevos algoritmos

Dada su filosofía de trabajo no permite aprovechar las soluciones encontradas por otras metaheurísticas, excepto la que resulta trivialmente al tomarlas como solución inicial.

Es aplicable la "paralelización trivial" mencionada en *"Consideraciones generales"* al comienzo de esta sección, donde:

- La que consta en instancias independientes se logra asignando un procesador independiente a cada instancia encargada de resolver el problema con determinados parámetros de temperatura inicial, final y decremento de la misma. Notar que cada instancia es independiente de las demás y que NO colaboran entre sí.
- La que apunta a la reducción de la complejidad en alguna instancia intermedia depende de la factibilidad de paralelizar el algoritmo que a partir de la solución actual genera la próxima solución a menor temperatura.
- El fraccionamiento del problema en subproblemas que se resolverán con Simulated Annealing depende del problema a resolver.

Asimismo, puede lograrse una paralelización un poco más inteligente haciendo un intercambio de datos experimentales de temperaturas entre procesadores, entre otras cosas.

Percepción general

Prácticamente no se tiene control del comportamiento de la heurística, ya que los parámetros de ajuste no tienen relación directa con la búsqueda.
Se dificulta el concepto de búsqueda.
Posee una búsqueda local miope.
La diversificación también es miope, lo que dificulta identificar las regiones recorridas y ajustar la exploración.
Uno de los puntos que entendemos significativo es la posibilidad que brinda de obtener resultados relevantes en los casos en los que no se tiene conocimientos suficientes de cómo atacar un determinado problema.

Metaheurística: GRASP

Claridad

Tiene una filosofía natural en la medida en que es un proceso iterativo en el que cada ciclo lleva adelante una serie de pasos que resultan intuitivos.
Posee una búsqueda local clara y ajustable así como una diversificación también clara y ajustable. Donde la búsqueda local esta encapsulada justamente dentro del procedimiento de Búsqueda Local y la diversificación dentro del Constructor utilizado.
Independientemente de cómo se implementen los procedimientos de Construcción y Búsqueda Local la filosofía de trabajo promulgada tiene una clara relación con el problema a resolver.

Documentación disponible

Diversidad: Media
Calidad: Alta

Antecedentes de aplicación al problema de confiabilidad

Diversidad: Baja
Calidad: Baja

Aplicabilidad y complejidad de implementación

Complejidad: Media
Aplicabilidad: Alta

Versatilidad para mejorar y/o generar nuevos algoritmos

Gracias a la propiedad de adaptatividad de la metaheurística GRASP, además de la aplicación de la "paralelización trivial" mencionada en *"Consideraciones generales"* al comienzo de esta sección, es posible obtener una implementación paralela más sofisticada, en la que las instancias paralelas colaboran entre sí.

Percepción general

GRASP se adapta muy bien a las necesidades de nuestro taller.

Metaheurística: Ant System

Claridad

Tiene una filosofía natural que esta basada en la cooperación de agentes simples en la búsqueda de una solución en conjunto.
La diversificación es clara y ajustable y esta contenida en el proceso que guía a un agente en el hallazgo de una solución factible.
Posee una búsqueda local diseminada en la heurística (no esta centralizada en un lugar puntual), sin embargo esta búsqueda no constituye una búsqueda local propiamente dicha.
Parece no haber una clara relación entre el problema y la mecánica de resolución.

Documentación disponible

Diversidad: Alta
Calidad: Media

Antecedentes de aplicación al problema de confiabilidad

Diversidad: Baja
Calidad: Baja

Aplicabilidad y complejidad de implementación

Complejidad: Alta (+)
Aplicabilidad: Media

Versatilidad para mejorar y/o generar nuevos algoritmos

Permitiría aprovechar soluciones dadas por otras metaheurísticas a través de la traducción de dichas soluciones en rastros de feromonas, lo cual puede llevarse a cabo tanto al comienzo de la ejecución como en etapas intermedias. No obstante no esta claro el hecho de que esta estrategia proporcione una eficaz contribución.

Permite implementar los tres tipos de "paralelización trivial" mencionados en *"Consideraciones generales"* al comienzo de esta sección, donde:

- La paralelización que utiliza instancias independientes puede lograrse el implementar varias colonias de hormigas que en paralelo intentan resolver el mismo problemas. En este caso surge algo potencialmente enriquecedor que es la colaboración entre las colonias, intercambiando rastros de feromona en determinados momentos (sean estos correspondientes al rastro de feromona existente en cada colonia o los que resultan de las mejores soluciones encontradas por cada colonia); este intercambio puede hacerse entre todas las colonias o solo entre algunas de ellas.
- La paralelización que divide o reduce la complejidad en alguna etapa se logra asignando a cada hormiga (agente) un procesador paralelo, debido a que se requiere el manejo de memoria compartida entre agentes es posible que frente a ciertos problemas esta paralelización no sea factible.

- Dependiendo del problema, es en principio factible fraccionarlo en subproblemas, cada uno de los cuales sería resuelto por una colonia independiente, luego se combinaría las soluciones parciales para encontrar la solución al problema original.

Percepción general

La intensificación está diseminada en la metaheurística no estando centralizada en un lugar puntual, por lo que en términos clásicos no se tiene una búsqueda local.

Metaheurística: Neural Networks

Claridad

A pesar de la naturalidad de la idea base de esta metaheurística, consistente en entrenar una cierta entidad para luego utilizarla en la determinación de la mejor solución, cuando es preciso hacer uso de la misma se manifiesta una complejidad adicional. Nos referimos a que en la etapa de entrenamiento no queda claro como se materializa el aprendizaje de la red y en muchos casos se pueden producir desajustes en el mismo (no esta asegurada la convergencia ni la monotonía del aprendizaje) si las situaciones manejadas en el proceso de entrenamiento no son las correctas.

Es prácticamente imposible imaginar el recorrido llevado a cabo a través del espacio de soluciones por una Neural Network mientras intenta encontrar la solución a un problema. Asimismo no es posible definir mecanismos claros y ajustables para controlar la diversificación e intensificación que realiza una Neural Network entrenada durante la búsqueda.

Documentación disponible

Diversidad: Baja
Calidad: Media

Antecedentes de aplicación al problema de confiabilidad

Diversidad: Baja (-)
Calidad: Baja (-)

Aplicabilidad y complejidad de implementación

Complejidad: Alta (+)
Aplicabilidad: Baja (-)

Versatilidad para mejorar y/o generar nuevos algoritmos

No permite la alternativa de complementar la búsqueda de la mejor solución con algoritmos basados en otras metaheurísticas.

Permite implementar los tres tipos de "paralelización trivial" mencionados en *"Consideraciones generales"* al comienzo de esta sección, donde:

- La paralelización que utiliza instancias independientes puede lograrse asignando una o más neuronas a procesadores independientes.
- La paralelización que divide o reduce la complejidad en alguna etapa se logra paralelizando el cálculo llevado a cabo por cada neurona, si este así lo requiere.
- Dependiendo del problema, es en principio factible fraccionarlo en subproblemas pero con una dificultad impuesta por la arquitectura de la red; que hace que una Neural Network particular sea aplicable solo a problemas de tamaño fijo.

Percepción general

Desde el punto de vista general encontramos las siguientes puntualizaciones:

- El aprendizaje no se materializa. No se refleja una acumulación de experiencia a lo largo del aprendizaje, lo que no transmite ninguna garantía de que se este haciendo efectivo el mismo.
- Existen problemas de convergencia en el aprendizaje, que suelen aparecer al hacer un sobreentrenamiento de la red. El problema aquí es que no se dispone de un indicador que permita determinar si se esta llegando a un determinado nivel de sobreentrenamiento de la red.
- Por los puntos anteriores y considerando que el aprendizaje se hace para un subconjunto particular de datos, nuevamente no existen garantías de que sea posible obtener soluciones de calidad media; y muchas veces tampoco de obtener soluciones que cumplan con las restricciones del problema.
- Se hace difícil encontrar una configuración que sea independiente del "tamaño" del problema, lo que la hace muy poco aplicable a problemas de grafos. Cosa que no sucede con otro tipo de problemas como por ejemplo el de reconocimiento de caracteres escritos a mano.

Por los puntos antes expuestos entendemos que esta metaheurística no es aplicable al problema objeto de nuestro taller.

4.3.2. Proceso de Selección

En esta sección presentamos, en forma simplificada, el proceso por el cual seleccionamos las dos metaheurísticas más adecuadas para el problema objetivo de este taller. Al presentarlo nos vemos obligados a estructurar y simplificar las discusiones que al respecto llevamos a cabo, discusiones que siguieron numerosos caminos y que por momentos se caracterizaron por tener marchas y contramarchas en la búsqueda del resultado deseado. Esta estructuración imposibilita describir el proceso que realmente seguimos para determinar las metaheurísticas más adecuadas; sin embargo hemos hecho los mayores esfuerzos para conjugar lo realmente sucedido con una redacción clara y fácil de entender.

El mecanismo de selección se basó en la idea de agrupar metaheurísticas equivalentes. Los grupos de metaheurísticas se ordenarían de acuerdo a la preferencia que las mismas les confirieran. Con este ordenamiento de grupos se seleccionarían finalmente las dos metaheurísticas más adecuadas.

El método que utilizamos para obtener esta lista de grupos fue el siguiente.

1- Comenzamos con un solo grupo de metaheurísticas al cual llamamos G1, dentro del cual incluimos todas las metaheurísticas consideradas; situación que puede interpretarse como el tener una lista de grupos con un solo elemento. De esta manera, en una primera instancia consideramos que todas las metaheurísticas son equivalentes en lo que a resultar seleccionadas se refiere.

2- Tomando individualmente cada criterio en el orden de prioridades asignado, hacemos lo siguiente:

- **a.** Aplicamos el criterio a cada grupo Gi de metaheurísticas determinando para cada uno de estos grupos los subgrupos de metaheurísticas equivalentes en lo que respecta al criterio que se está aplicando.
- **b.** Numeramos los subgrupos identificados en **a** obteniendo los subgrupos Gi.1,...,Gi.n de modo que todas las metaheurísticas del subgrupo Gi.j estén en un nivel de valoración mayor respecto del criterio aplicado, que las del subgrupo Gi.k, siendo k>j.
- **c.** Sustituimos el grupo Gi, en la lista de grupos, por los subgrupos que se numeraron en **b**. Esta sustitución se hace manteniendo el orden de los subgrupos obtenido en **b**.
- **d.** Renumeramos completamente la nueva secuencia de grupos comenzando desde el número 1, respetando el orden de la secuencia obtenida hasta el momento.
- **e.** Volvemos a aplicar todo el proceso a partir del punto **a** para el siguiente criterio si lo hubiera.

Etapa 1:
Al comienzo la situación es la siguiente:
u

G1
Ant Systems
Algoritmos Genéticos
GRASP
Neural Networks
Simulated Annealing
Tabu Search

Esto representa la situación inicial en la que se desconoce el resultado de la aplicación del proceso de selección.

Etapa 2: Aplicación del criterio *Claridad*
A partir de la situación obtenida en la Etapa 1 y luego de la aplicación de los pasos **a** y **b** resultan los siguientes subgrupos:

G1.1
Algoritmos Genéticos
GRASP
Tabu Search

G1.2
Ant Systems

G1.3
Neural Networks
Simulated Annealing

Esta situación se interpreta de la siguiente manera: las metaheurísticas incluidas en el subgrupo G1.1 son más adecuadas (hasta el momento) que las pertenecientes al subgrupo G1.2; las metaheurísticas pertenecientes a este último son a su vez más adecuadas para los objetivos de este taller que las pertenecientes al subgrupo G1.3.

Finalizada la aplicación de este criterio el orden de grupos obtenidos es el siguiente:

G1	G2	G3
Algoritmos Genéticos	Ant Systems	Neural Networks
GRASP	----------	Simulated Annealing
Tabu Search	----------	----------

Etapa 3: Aplicación del criterio *Documentación disponible*
Este criterio involucra dos características asociadas a la documentación disponible para cada metaheurística, nos referimos a la *calidad* y a la *diversidad* de la información disponible. En términos de valoración relativa no es posible afirmar que una de ellas tiene mayor peso que la otra, por lo que al aplicar este criterio asignamos igual peso a las dos.

A los efectos de saber como valora este criterio cada metaheurística se hizo uso de la siguiente escala:

Valor	Puntaje
Baja (-)	1
Baja	2
Baja (+)	3
Media (-)	4
Media	5
Media (+)	6
Alta (-)	7
Alta	8
Alta (+)	9

Posteriormente esta escala fue usada en criterios que realizan la valoración de forma similar.

Combinando los valores de calidad y diversidad de la documentación disponible para cada metaheurística, se agruparon las mismas en conjuntos de equivalencia. El que dos metaheurísticas pertenezcan al mismo conjunto de equivalencia se interpreta como que según el criterio "Documentación disponible" ambas tienen la misma valoración.

Conjunto 1: (17 puntos)
Algoritmos Genéticos

Conjunto 2: (13 puntos)
GRASP
Ant Systems

Conjunto 3: (9 puntos)
Tabu Search

Conjunto 4: (7 puntos)
Simulated Annealing
Neural Networks

Luego de aplicados los pasos 2.a y 2.b resultan:

G1.1	G1.2	G1.3
Algoritmos Genéticos	GRASP	Tabu Search

G2
Ant Systems

G3
Neural Networks
Simulated Annealing

Obteniéndose el orden de grupos siguiente:

G1	G2	G3	G4	G5
Algoritmos Genéticos	GRASP	Tabu Search	Ant Systems	Neural Networks
----------	----------	----------	----------	Simulated Annealing

Etapa 4: Aplicación simultánea de los criterios *Antecedentes de aplicación al problema de confiabilidad en redes* y *Aplicabilidad y complejidad de implementación*.

El criterio *Antecedentes de aplicación al problema de confiabilidad en redes* tiene asociadas *calidad* y *diversidad,* de igual manera que *Documentación disponible*. A los efectos de la valoración se utilizó la misma mecánica.

En lo que respecta a *Aplicabilidad y complejidad de implementación* se esta frente a una situación similar, pero en la que se debe tener en cuenta que la *Complejidad* tiene un sentido opuesto al de *Aplicabilidad.* Por tanto se obtendrá el valor final a partir de la diferencia de los mismos.

Utilizando el mecanismo de sumar puntos para estas metaheurísticas tenemos los siguientes resultados:

Conjunto 1: (10 puntos)
Algoritmos Genéticos

Conjunto 2: (7 puntos)
GRASP

Conjunto 3: (4 puntos)
Tabú Search

Conjunto 4: (0 puntos)
Ant System

Conjunto 5: (-3 puntos)
Simulated Annealing

Conjunto 6: (-6 puntos)
Neural Networks

A partir de estos conjuntos, de los grupos identificados finalizada la etapa 3 y luego de ejecutar los pasos 2.a y 2.b del método para seleccionar las metaheurísticas se determinan los siguientes subgrupos:

G1	G2	G3	G4
Algoritmos Genéticos	GRASP	Tabu Search	Ant Systems

G5.1	G5.2
Simulated Annealing	Neural Networks

Aplicando los pasos 2.c y 2.d del método de selección...

G1	G2	G3	G4	G5	G6
Algoritmos Genéticos	GRASP	Tabu Search	Ant Systems	Simulated Annealing	Neural Networks

Etapa Final : Aplicación del criterio *Versatilidad* y *Percepción General*
Finalizada la etapa 4 se ha obtenido un ordenamiento de las metaheurísticas para el que no es posible obtener nuevos refinamientos. Ya no es necesario entonces continuar con el proceso, quedando sin aplicar los criterios *Versatilidad* y *Percepción General*. De todas maneras, si se lleva a cabo una lectura cuidadosa de los puntos destacados bajo estos criterios se puede observar que apoyan el orden obtenido.

4.3.3. Resultado Final

Sobre la base de lo expuesto en la sección anterior, puntualizamos entonces que las metaheurísticas más adecuadas para el problema objetivo de este taller son, en orden de preferencia:

1. Algoritmos Genéticos.
2. GRASP.
3. Tabu Search.
4. Ant Systems.
5. Simulated Annealing.
6. Neural Networks.

Debido a que para la implementación de los algoritmos que resolverán nuestro problema debemos seleccionar las dos más adecuadas, concluimos que, los mismos se llevarán a cabo sobre la base de los principios promulgados por Algoritmos Genéticos y GRASP.

Capítulo 5
Algoritmos Implementados

5. Algoritmos Implementados

En esta sección presentamos los algoritmos que hemos desarrollado. Para cada uno de los mismos presentaremos una breve introducción y a continuación una especificación del mismo a nivel de pseudocódigo.

Notar que no todas las puntualizaciones relevantes están presentes en la introducción correspondiente. Por el contrario, la mayoría de ellas se encuentran embebidas dentro de los pseudocódigos, por ser ese el lugar más adecuado para las aclaraciones o comentarios que hacen referencia a aspectos puntuales de los algoritmos presentados.

Los algoritmos han sido implementados en el lenguaje de programación C++ y tienen como base algunos puntos que nosotros definimos frente a las características propias del problema objetivo:

- La solución debe estar encaminada a la creación de caminos simples que comuniquen s con t. Esto es, fijamos el objetivo de búsqueda de caminos de s a t y no la construcción directa de topologías adecuadas, lo que parecía ser la idea en un principio más clara y por momentos la única.
- La solución final debe estar formada por un subconjunto de caminos cuya confiabilidad según el costo en conjunto sea la mejor. Teniendo un conjunto finito de los mejores caminos no sería en principio difícil entonces construir la solución final. Véase sin embargo, que no es un problema sencillo computacionalmente el encontrar el subconjunto del conjunto anterior de caminos mejores que maximice la confiabilidad.
- Asignar dinámicamente la mayor cantidad de parámetros que controlan el comportamiento de los algoritmos implementados, en función de ciertas características de la instancia del problema que se intenta resolver. Esto apunta a no dejar esta asignación de valores críticos a cargo de un usuario, quién no tendría una idea muy acertada de como impactarían los mismos en el comportamiento de los algoritmos y en la calidad de las soluciones.

A continuación incluimos algunas decisiones y recomendaciones generales que entendimos facilitarían la comprensión de los pseudocódigos que se incluyen en este capítulo.

Decisiones y Recomendaciones Generales:

- En general recomendamos que se lea con algo de detenimiento los nombres, parámetros y descripciones de las funciones, ya que los mismos han sido elegidos de forma cuidadosa (esto no significa que no existan errores ni mejoras que pudieron habérsele hecho) además de que la información que brindan está condensada. Una lectura ligera puede hacer perder detalles esclarecedores y ampliadores de la funcionalidad de los mismos.
  En lo que refiere al nivel elegido para la realización del pseudocódigo, entendimos que no era necesario la realización de pseudocódigo para algunas funciones en particular. Teniendo presente que el objetivo inicial de los pseudocódigos era el brindar una idea algorítmica primaria de la solución, nos llevó a dejar funciones sin detalle, ya que, por sus características, con el comentario o el nombre asociado a las mismas ya resultaban auto explicativo. Al mismo tiempo se han dejado funciones sin pseudocódigos, pues su realización estaría íntimamente ligada a las decisiones de implementación que se llevarían a cabo en etapas posteriores.

- En lo que refiere a la diferencia en el estilo de inclusión de comentarios en los pseudocódigos y códigos fuentes, hemos tomado algunas consideraciones personales.

  Los comentarios incluidos en los pseudocódigos pueden verse como un "anexo" en la medida que puede resultar más claro leer el pseudocódigo mismo antes que el comentario. Por tal motivo los mismos han sido indicados a continuación de la instrucción e indentados con respecto a la misma.

```
instrucción;
        //comentario sobre la instrucción o instrucciones
```

  No ocurre lo mismo con los comentarios realizados al código (incluidos en Capítulo 9 "Apéndices") los cuales seguramente serán referenciados en primer lugar, dejando de lado la codificación propiamente dicha. Por esta razón han sido incluidos inmediatamente antes y alineados con respecto a la instrucción.

```
//comentario sobre la instrucción o instrucciones
instrucción;
```

5.1. Algoritmo de Simplificación

5.1.1. Introducción

Como todo problema a resolver, el que debemos atacar debe ser modelado de alguna manera para luego encontrar la solución operando sobre este modelo.

Analizando el problema planteado identificamos algunas características sutiles, que permitirían simplificar la instancia del problema y que en un comienzo no fueron fácilmente identificables.

Dado que nuestro problema consiste en el diseño de redes Diámetro-Confiables, la idea básica de nuestro proceso de simplificación fue la de eliminar de la instancia inicial aquellos nodos para los que era posible determinar a priori que nunca formarían parte de la solución al problema. Dado que se tiene un parámetro de distancia, que acota el largo de los caminos de s hacia t, notamos que era posible eliminar los nodos, para los que, el largo de los caminos más cortos que pasaban por ellos y comunicaban s con t era mayor que la distancia D. Luego de esa eliminación se llevaría a cabo una eliminación de los nodos que no fueran los nodos terminales s y t y que tuvieran grado menor o igual a uno. También se llevaría a cabo algunas consideraciones obvias relativa a la imposibilidad de incluir aristas en la solución final por tener confiabilidad cero o costo mayor que el presupuesto.

La lógica del procedimiento de Simplificación que implementamos consiste en lo siguiente:

- Aplicar Dijkstra desde s y asignar a cada vértice v del grafo el valor Ds(v) que representa el largo del camino más corto de s a v.
- Aplicar Dijkstra desde t y asignar a cada vértice v del grafo el valor Dt(v) que representa el largo del camino más corto de t a v.
- Eliminar del grafo todos los vértices para los que se cumpla que Ds(v)+Dt(v)>D, es decir, todos los vértices para los que el camino más corto que pasa por ellos y que va de s a t tiene largo mayor que D.
- Mientras haya vértices, distintos a s y t, de grado menor o igual a 1 se eliminan del grafo.

La idea que manejamos se materializó en el siguiente enunciado.

Primer enunciado del Teorema de Simplificación

Sea G', la red resultado de aplicar el algoritmo de Simplificación sobre la red G.
Para toda red G, fuente s, terminal t y para todo diámetro D se cumple que
$DiamConf_{st}(G,D)=DiamConf_{st}(G',D)$.

Segundo enunciado del Teorema de Simplificación

Sea G', la red resultado de aplicar el algoritmo de Simplificación sobre la red G.
Es equivalente resolver el problema de encontrar la subred que maximice la D-confiabilidad fuente-terminal sujeto a restricciones de costo en las redes G y G'.

Corolario: Aplicabilidad de la Simplificación

El procedimiento propuesto es llevado a cabo sobre la base del largo de los caminos. El mismo puede ser aplicado de forma no excluyente sobre los costos de los caminos, de forma tal de eliminar los nodos para los que los caminos menos costosos que pasan por ellos y comunican s con t tienen un costo mayor que el parámetro K.

Corolario: Generalización de la Simplificación

Sea G', la red resultado de aplicar el algoritmo de Simplificación sobre la red G.
Para toda red G con K terminales y para todo diámetro D se cumple que $DiamConf_K(G,D) = DiamConf_K(G',D)$.

5.1.2. Pseudocódigo

```
/++++++++++++++++++++++++++++++++++++++++++++++++++/
/+                                                +/
/+                  Simplificación                +/
/+                                                +/
/++++++++++++++++++++++++++++++++++++++++++++++++++/
```

Algoritmo de Simplificación:

```
FUNCION Simplificación();
        G'= G;
        EliminarAutoaristas(G');

        Ds=Dijkstra(G', s);
                // la celda Ds[i] tiene el largo del camino más corto de s al vértice i.

        Dt=Dijkstra(G', t)
                // la celda Dt[i] tiene el largo del camino más corto de i al vértice t.

        PARA CADA vértice i de G' distinto de s y t HACER
                // verifico si el camino más corto que une a s y  t pasando por i es más largo que D
                // si es lo elimino del Grafo .

                SI  Ds[i]+Dt[i] > D ENTONCES
                        Eliminar_vertice(G',i);
                FIN SI
        FIN PARA CADA

        MIENTRAS ExistanVerticesDeGrado1(G') HACER
                PARA CADA vértice i de G' distinto de s  y t de grado 1 HACER
                        Eliminar_vertice(G',i);
                FIN PARA CADA
        FIN MIENTRAS

        RETORNAR (G');
FIN FUNCION Simplificacion();
```

5.2. Algoritmo GRASP

5.2.1. Introducción

La primera solución realizada para el problema objetivo fue sobre la base de los principios promulgados por la metaheurística GRASP. Posteriormente llevamos a cabo otro algoritmo sobre la base de los principios que sugiere Algoritmos Genéticos, el que heredó la mayoría de los fundamentos básicos dados a GRASP, ganando así la comparación más efectiva de ambas metaheurísticas.

Durante su procesamiento, el algoritmo desarrollado sobre la base de la metaheurística GRASP, divide el presupuesto de trabajo en una fracción para el Constructor y otra para la Búsqueda Local. La asignación de dichas fracciones de presupuesto se realiza de forma aleatoria, dándole un mayor peso a aquellas que históricamente han demostrado ser las más convenientes. Esto último resume la adaptabilidad implementada para GRASP.

Al comenzar la fase de Construcción se subdivide el presupuesto asignado según un parámetro dado, el cual determina el número de fases de trabajo para el Constructor.

Durante cada fase se construyen caminos hasta completar el presupuesto asignado a ésta. El total de caminos construidos, denominados caminos explícitos, son identificados para su posterior uso.

Definimos criterios para determinar cuando el presupuesto de una fase, o lo que resta del mismo, no permitiría agregar caminos; en cuyo caso, dicho resto se acumula a la fracción de presupuesto que se asignará a la siguiente fase.

La construcción de caminos se realiza partiendo del vértice s y en cada instancia se sortea cual será el próximo vértice que formará parte del camino, evitando considerar aquellos ya incluidos en éste. Dicha elección se realiza de forma aleatoria, valorando la confiabilidad de las aristas intermedias y sin considerar aquellas aristas que durante la fase actual fueron definidas como no usables.

Durante la construcción de un camino se considera además la restricción impuesta para el largo. Si en algún instante dicha restricción es superada, se intenta realizar un atajo que alcance el vértice terminal t. Este atajo surge considerando todas las posibles aristas que comunican alguno de los vértices del camino en construcción con el vértice t. De los posibles atajos de esta forma planteados, se elige aquel que resulte en el camino s-t de mayor confiabilidad.

A la solución obtenida por el Constructor GRASP se le aplica un mejoramiento encapsulado en el procedimiento de Búsqueda Local. Este mejoramiento se basa en la identificación, selección y posterior aplicación de los posibles exchanges entre pares de caminos. Entiéndase como acepción par el término “exchange” la usual en Teoría de Grafos.

El proceso de búsqueda local se lleva a cabo en etapas sucesivas, en cada una de las cuales se selecciona un conjunto de pares de caminos del total de caminos

explícitos presentes en el grafo actual. Posteriormente se listan los exchanges posibles entre cada par de caminos propuestos. Su identificación se realiza considerando todos los pares posibles de aristas entre los dos caminos del par, evaluándose si cada uno de ellos resultaría beneficioso en algún sentido.

Los exchanges identificados de esta manera son ordenados convenientemente. Se asigna una mayor importancia a aquellos que mejoran la confiabilidad de los nuevos caminos explícitos que surgirían de su aplicación y que simultáneamente disminuyen el costo de la solución. También son considerados con una menor importancia aquellos que sólo mejoran la confiabilidad de los nuevos caminos y a aquellos que solo disminuyen el costo de la solución.

Cada etapa de la Búsqueda Local culmina cuando se realiza la aplicación de los intercambios propuestos. Esta aplicación se lleva a cabo sobre un grafo temporal a los efectos de poder determinar que se ha mejorado la solución en esa etapa. Aún en los casos en los que una etapa anterior haya fracasado en el mejoramiento, siempre tiene sentido comenzar una nueva, debido a que la generación de los pares de caminos se realiza de forma aleatoria.

5.2.2. Pseudocódigo

```
/++++++++++++++++++++++++++++++++++++++++++++++++++++++/
/+                                                    +/
/+                      GRASP                         +/
/+                                                    +/
/++++++++++++++++++++++++++++++++++++++++++++++++++++++/

/+-----------------
```

Algoritmo Principal

```
                                                  -----------------+/
```

```
FUNCION GRASP(GOriginal *I*, TopeIteraciones *I*,
                                    D *I*, K *I*, GSolucionGrasp *O*);
```

Parámetros:

I - GOriginal:	Grafo asociado al problema original que se debe resolver.
I - TopeIteraciones	Número máximo de iteraciones que se quieren llevar a cabo.
I - D:	Parámetro del problema original que especifica la restricción para el largo de los caminos s-t.
I - K:	Restricción de costo para el problema original.
O - GSolucionGrasp:	Grafo construido por este algoritmo.

```
GSolucionGrasp =G({s,t},{});
    //Grafo que únicamente tiene los vértices s y t, para este grafo la confiabilidad vale cero.

GOriginal=Simplificacion(GOriginal);

TopeIntentosSucesivosFallidosDeMejora
    //Tolerancia máxima (medida en intentos sucesivos) de fallos al intentar mejorar la soluciòn
    //actual de GRASP. Se utiliza como criterio de parada para indicar cuando se estima que se //ha
    alcanzado la solución que puede ser la óptima.

IntentosSucesivosFallidosDeMejora=0;

TopeIteraciones
    //Tope máximo de iteraciones a llevar a cabo en GRASP

Iteraciones=0;

MIENTRAS SeDebaContinuar(Iteraciones, TopeIteraciones,
                                    IntentosSucesivosFallidosDeMejora) HACER
    GConstructor=ConstructorGrasp(GOriginal, Diametro, K,
                                                    NroDeMejorasSucesivas);
    GBusquedaLocal=BusquedaLocalGrasp(GConstructor, Diametro, Presupuesto);
        //Heurística para decidir el Cálculo de la Confiabilidad: Dependiendo del costo de Calcular
        //la Confiabilidad se puede definir un criterio que vincule la confiabilidad de los caminos
        //entre las soluciones y decida si hay probabilidad de que se cumpla la supremacía de la
        //nueva solución con respecto a la actual.
```

```
        SI (GBusquedaLocal.Confiabilidad() > GSolucionGrasp.Confiabilidad())
                                                                ENTONCES
            IntentosSucesivosFallidosDeMejora=0;
            GSolucionGrasp=GBusquedaLocal;
        SINO
            IntentosSucesivosFallidosDeMejora++;
        FIN SI
    FIN MIENTRAS
FIN FUNCION GRASP
```

```
/+------------------
                    Algoritmo del Constructor
                                    ------------------+/
```

```
FUNCION ConstructorGrasp(GOriginal *I*, Diametro *I*, K *I*,
                         NroDeMejoresSucesivas *I*,  GConstructor *O*);
```

Parámetros:

I - GOriginal:	Grafo asociado al problema original que se debe resolver.
I - Diametro:	Parámetro del problema original que especifica la restricción para el largo de los caminos s-t.
I - K:	Restricción de costo para el problema original.
I - NroDeMejoresSucesivas:	Número de fases que se quieren llevar a cabo para encontrar caminos solapados que formarán parte del grafo construido por esta función.
O - GConstructor:	Grafo construido por esta función.

```
// Construyo un grafo GConstructor



GConstructorTemporal={};
    //Grafo temporal generado en cada iteración

GConstructor={};
    //Grafo final resultado de la superposición de Grafos temporales generados
    // en cada iteración GconstructorTemporal.

AristasAConsiderar=E;
    //Inicializo especial de aristas que serán consideradas durante la construcción de GConstructorTemporal
    //Pasible a nuevos ajustes para ejecución de pruebas: considerar (subconjunto, mejores aristas E

KTotalUsado=0;
    //Suma total de costos utilizados **holguras utilizadas** en cada iteración.

KParcialUsado=0;
    //Costo utilizado en una iteración en particular.
```

```
KFraccion= K /(NroDeMejorasSucesivas);
        //Fracción de "presupuesto" que se puede utilizar en cada iteración para construir el grafo temporal
        //GconstructorTemporal.

KActual=KFraccion;
        //Máxima holgura en costo que puede considerarse en cada iteración para la construcción del grafo temporal
        //GconstructorTemporal

KRestriccion=KFraccion;
        //Valor mínimo de holgura necesario que requiere la función
        //ConstruyoGrafoDeCaminosFactiblesS-T() para trabajar

EsPosibleContinuar=True;
        // Bandera utilizada para controlar las iteraciones en el Constructor.

SeConstruyoGrafoDeCaminos=?;
        //Bandera utilizada para indicar si se ha construido o no un grafo GConstructorTemporal
        //en la iteración actual

MIENTRAS EsPosibleContinuar HACER
        SeConstruyoGrafoDeCaminos=
                        ConstruyoNuevoGrafoDeCaminosFactiblesS-T( KActual *I*,
                                GConstructorTemporal *I/O*,
                                AristasAConsiderar *I/O*,
                                KParcialUsado *O*);
        SI SeConstruyoGrafoDeCaminos ENTONCES
                GConstructor= GConstructor UNION GConstructorTemporal;
                        // actualizo el Grafo General que estoy construyendo , manteniendo identificados
                        //y ordenados por confiabilidad los caminos pues serán utilizados posteriormente

                KTotalUsado= KTotalUsado + KParcialUsado;
                        // actualizo el total de holgura con lo utilizado en KParcialUsado
        FIN SI


        ActualizarKActual (K *I*, SeConstruyoGrafoDeCaminos *I*, KFraccion *I*,
                        KParcialUsado *I*, KTotalUsado *I/O*, KActual *I/O*,
                        KRestriccion *I/O*, EsPosibleContinuar *O*);
                //Actualizo en KActual el valor de la holgura para el paso siguiente, holgura que
                // será considerada por la función ConstruyoGrafoDeCaminosFactiblesS-T; esta
                // actualización tiene la restricción de Krestricción que es el valor mínimo aceptable
                // para que dicha función pueda trabajar.

                //Si SeConstruyoGrafoDeCaminos entonces se reinicializa el valor de KActual a KRestricción.
                //Si no actualizo KActual aumentando la holgura de trabajo pues no fue posible generar
                // caminos de s-t con la holgura considerada.

FIN MIENTRAS
FIN FUNCION ConstructorGrasp
```

```
FUNCION ActualizarKActual (K  *I*, SeConstruyoGrafoDeCaminos *I*, KFraccion *I*,
                           KParcialUsado *I*, KTotalUsado *I*,  KActual *I/O*,
                                KRestriccion *I/O*, EsPosibleContinuar *O*)
```

Parámetros:

I - K :	Restricción de Costo para el Problema.
I - SeConstruyoGrafoDeCaminos:	Bandera utilizada para indicar si se ha construido o no un grafo GConstructorTemporal. En esta función se utiliza para inicializar o incrementar KActual
I - KFraccion :	Fracción de "presupuesto" que se utiliza a los efectos de actualizar el KActual
I - KParcialUsado:	Costo utilizado en la última iteración
I/O - KTotalUsado:	Suma total de costos utilizados hasta el momento
I/O - KActual :	Holgura a ser actualizada en esta función
I/O - KRestriccion :	Valor mínimo de holgura necesario para inicializar KActual
O - EsPosibleContinuar :	Bandera utilizada para indicar que no fue posible actualizar KActual

```
SI  SeConstruyoGrafoDeCaminos ENTONCES
    // actualizo KActual que es la holgura que será considerada  por la función
    // ConstruyoGrafoDeCaminosFactiblesS-T, esta actualización tiene la restricción de
    // Krestricción que es el valor mínimo aceptable para que dicha función pueda  trabajar

    SI K - KTotalUsado >= KRestriccion ENTONCES
        KActual=KRestriccion;
        // verifico que el próximo paso tenga por lo menos KRestriccion de holgura
        // para trabajar. Si no tiene agrego dicha holgura a  KActual

        SI K - KTotalUsado-KActual < KRestriccion ENTONCES
            KActual=KActual + (K - KTotalUsado-KActual);
        FIN SI
    SINO
        // No puedo considerar la diferencia que resta  K-KTotalUsado aunque
        //sea mayor que cero pues es menor que KRestriccion
        //  Marco como que no puedo seguir

        EsPosibleContinuar=False;
    FIN SI
SINO
// Si no SeConstruyoGrafoDeCaminos entonces tengo que aumentar la cota KActual con la que estoy trabajando.
// Para ello estudio la posibilidad que tengo de asignar dicho valor

    SI K - KTotalUsado  >= (KFraccion + KActual) ENTONCES
        KActual=KActual + KFraccion;
        // Actualizo la holgura mínima necesaria para que la función
        // ConstruyoGrafoDeCaminosFactiblesS-T pueda trabajar actualizo entonces
        // KRestriccion.

        KRestriccion=KActual;
        //Verifico que el próximo paso tenga por lo menos KRestriccion de holgura
        // para trabajar. Si no tiene agrego dicha holgura a  KActual.

        SI (K-KTotalUsado-KActual) < KRestriccion ENTONCES
            KActual=KActual + (K - KTotalUsado-KActual);
        FIN SI
```

```
            SINO
                SI (K-KTotalUsado-KActual) >0 ENTONCES
                    // sino puedo aumentar en KFraccion aumento en  K-KTotalUsado-KActual.

                    KActual=KActual +(K - KTotalUsado -KActual);
                SINO
                    //Como no puedo agregar ninguna holgura adicional a KActual no tiene sentido
                    // volver a intentar utilizar la función ConstruyoGrafoDeCaminosFactiblesS-T por lo
                    // que marco como que no es posible continuar

                    EsPosibleContinuar=False;
                FIN SI
            FIN SI
        FIN SI
FIN FUNCION ActualizarKactual
```

```
FUNCION ConstruyoNuevoGrafoDeCaminosFactiblesS-T( KActual *I*,
        GconstructorTemporal *I/O*, AristasAConsiderar *I/O*, KParcialUsado *O*)
```

Parámetros:

I - KActual :	Holgura a ser utilizada en esta función
I/O -GConstructorTemporal:	Grafo temporal generado en esta función
I/O - Aristas Aconsiderar:	Conjunto especial de aristas que serán consideradas durante la construcción de GConstructorTemporal
O - KParcialUsado:	Costo utilizado en esta invocación

```
ConjuntoDeAristasUsadasPorVezPrimeraEnCaminoActual={};
    //Son las aristas que formando parte del CaminoActual es la primera vez que se usan.

EsPosibleContinuar=True;
    // Bandera utilizada para controlar las iteraciones en el Constructor

KParcialUsado=0;
    //Costo utilizado en una iteración en particular.

VerticeActual=s;
    //Variable que contiene el vértice extremo final del camino que se esta construyendo. Recordar que
    //el extremo inicial es el vértice s.

SucesivosIntentosFallidos=0;
    //Contador del número de sucesivos intentos fallidos al armar caminos de s a t.

ToleranciaDeSucesivosIntentosFallidos
    //Cota superior para el número de sucesivos intentos fallidos al armar caminos de s a t.

ToleranciaDeSucesivosIntentosDeUso
    //Cota superior para el número de sucesivos intentos de uso de las aristas al armar caminos de s a t.

InicializoComoNoUsadas(AristasAconsiderar);
    //A cada arista del conjunto AristasAConsiderar se le inicializa en False al atributo Usada. Para cada arista este
    //atributo es True si y solo si la arista forma parte de un camino s-t construido por el constructor.
```

```
CaminioActual={};
        //Se inicializa como el camino vacío.

MIENTRAS (KParcialUsado <KActual) Y (EsPosibleContinuar) HACER
        PudeCostoYLargo=ElijoArista(CaminoActual, VerticeActual,AristaElegida);
                //Se selecciona una arista que a partir de VerticeActual continúe el camino que se esta armando.
                //La elección de esta arista se hace teniendo en cuenta las aristas que pueden continuar el camino
                //a partir de VerticeAcutal, de las que se sortea al azar una de ellas


                //CaminoActual es usado para evitar repetir vértices

                //A los efectos de evitar que el algoritmo no considere en forma repetitiva un mismo camino (que ya esta
                // descubierto o que no se llega a armar) se decidió utilizar cotas en el uso de las aristas, coaccionando
                //la investigación en otros sectores del grafo. Esto se lleva a cabo haciendo que cada arista tenga
                //asociada un atributo llamado SucesivosIntentosDeUso que acumula el número de veces que se intentó
                //sin éxito utilizar esa arista en caminos que finalmente no fueron armados. Si ese número llega al
                //TopeSucesivosIntentosDeUso dicha arista se dejará de considerar indefinidamente en la fase.

                //Asimismo se utiliza FraccionTopeSucesivosIntentosDeUso (que corresponde a una fracción de la cota
                //TopeSucesivosIntentosDeUso) como primer cota en el uso de aristas a los efectos de evitar inutilizar
                //las mismas por alcanzar tempranamente la cota fijada. El alcance de forma temprana sería
                //ocasionado por la ponderación desmedida de dichas aristas que ocasionan la selección sistemática de
                // las mismas. Esta cota se usaría como indicador para disparar la corrección de la ponderación dada en
                //el algoritmo de selección, haciendo que por ejemplo dichas aristas (que alcanzaron
                //FraccionTopeSucesivosIntentosDeUso) pierdan la ponderación asociada dejándola con igual
                //probabilidad que otras. De esta manera el tope no sería agotado en ese proceso cíclico que no brinda
                //beneficio alguno.

                //En el caso de que se esta armando un camino en el que se ha incluido una arista cuyo atributo Usada
                //valía False es posible considerar como factibles para el camino aristas cuyo atributo Usada es True a
                //pesar de que para las mismas se haya alcanzado el TopeSucesivosIntentosDeUso. No se consideran
                //aristas cuyo atributo Usada sea False que hayan alcanzado el TopeSucesivosIntentosDeUso pues
                //este contador revela características no deseables de la arista o de su entorno.

        SI PudeCostoYLargo ENTONCES
                //Actualizo KParcialUsado, VerticeActual y CaminoActual; y en caso de ser necesario, inicializo la
                // propiedad de Usada de la arista como True e incluyo la misma en el conjunto
                //ConjuntoDeAristasUsadasPorVezPrimeraEnCaminoActual.

                SI AristaElegida.Usada=False ENTONCES
                        KParcialUsado+=AristaElegida.costo;
                        AristaElegida.Usada=True;
                                // La considero con costo cero en el futuro.

                        ConjuntoDeAristasUsadasPorVezPrimeraEnCaminoActual U=
                                                                {AristaElegida};
                FIN SI
                VerticeActual=AristaElegida.OtroExtremo(VerticeActual);
                AgregoNuevaArista(CaminoActual,AristaElegida);

                SI AristaElegida.extremo == t ENTONCES
                        //Acabo de terminar con el armado del camino.

                        SI YaExisteEnEstaFase(CaminoActual) ENTONCES
                                //Realizada comoConjuntoDeAristasUsadasPorVezPrimeraEnCaminoActual={}
                                //lo considero como un fracaso y actualizo SucesivosIntentosDeUso
                                //para cada una de las aristas del camino
                                SucesivosIntentosFallidos++;
```

```
                    IncrementoSucesivosIntentosDeUsoParaCadaArista(
                                                        CaminoActual);
                SINO
                    SucesivosIntentosFallidos=0;
                        //Inicializo nuevamente el contador SucesivosIntentosFallidos

                    InicializoSucesivosIntentosDeUsoParaCadaArista(
                                                        CaminoActual);
                FIN SI
                VerticeActual=s;
                    //Si ya termine con el armado del camino, inicializo VerticeActual como el comienzo
                    //del nuevo camino

                ConjuntoDeAristasUsadasPorVezPrimeraEnCaminoActual={};
                    //Inicializo estructura de trabajo.

                ReinicializarLista(CaminoActual);
                    //Inicializo estructura de trabajo.
            FIN SI
        SINO //si no PudeCostoYLargo entonces ...
            EncontreUnAtajo=IntentoUnAtajo(CaminoActual,
                                ConjuntoDeAristasUsadasPorVezPrimeraEnCaminoActual,
                                                        CostoDiferencia);
                //Intento un atajo para llegar a t usando parte del camino armado hasta el momento. Este
                //atajo se arma con un fragmento inicial del camino que se estaba construyendo al que se le
                //anexa una arista para comunicarlo con t y de esta manera "completar" el camino.

                //intento un atajo para llegar a t usando parte del camino armado hasta el momento
                //CostoDiferencia es un número positivo o negativo que indica la diferencia entre el costo de la
                // arista agregada y las no consideradas que pertenecen al conjunto
                ///ConjuntoDeAristasUsadasPorVezPrimeraEnCaminoActual.

                //Debe inicializarse a False el atributo  Usada de las aristas que pertenecían a
                //ConjuntoDeAristasUsadasPorVezPrimeraEnCaminoActual que no forman parte del atajo
                // construido.

                //Se actualiza el conjunto ConjuntoDeAristasUsadasPorVezPrimeraEnCaminoActual
                //anexándole la arista agregada (en caso de que Usada=False) y quitándole las aristas no
                //consideradas en caso de que pertenecieran al mismo.

                //Nota: Al momento de construir el atajo puede hacerse de varias maneras: de los vértices que
                //se pueden conectar con t sorteo uno; de las aristas que me conectan con t elijo la mejor en
                //costo o conf o conf/costo. Tener en cuenta división por cero en caso de conf/costo.

            SI EncontreUnAtajo ENTONCES
                //Como ya terminé con el armado del camino, inicializo VerticeActual como el comienzo
                //del nuevo camino y actualizo KParcialUsado con el número CostoDiferencia
                //resultado de la diferencia entre la arista insertada y las no consideradas para
                //realizar el atajo

                KParcialUsado+=CostoDiferencia;

                //En este punto acabo de terminar con el armado del camino
```

```
SI YaExisteEnEstaFase(CaminoActual) ENTONCES
        //La condción de este if se implementa como
        // ConjuntoDeAristasUsadasPorVezPrimeraEnCaminoActual={}
        //Si se cumple considero como un fracaso y actualizo SucesivosIntentosDeUso
        //para cada una de las aristas del camino

        SucesivosIntentosFallidos++;
        IncrementoSucesivosIntentosDeUsoParaCadaArista(
                                                CaminoActual);
        SI (SucesivosIntentosFallidos >
                ToleranciaDeSucesivosIntentosFallidos) ENTONCES
                EsPosibleContinuar=FALSE;
        FIN SI
SINO
        SucesivosIntentosFallidos=0;
                //Reinicializo el contador SucesivosIntentosFallidos
        ReinicializoSucesivosIntentosDeUsoParaCadaArista(
                                                CaminoActual);
FIN SI
VerticeActual=s;
        //Si ya termine con el armado del camino, inicializo VerticeActual como el comienzo
        //del nuevo camino

ConjuntoDeAristasUsadasPorVezPrimeraEnCaminoActual={};
        //Inicializo estructura de trabajo.
ReinicializarLista(CaminoActual);
        //Inicializo estructura de trabajo.
SINO // Si no EncontreUnAtajo entonces...
        //Vuelvo a un estado que permita intentar la construcción de nuevos caminos...
        VerticeActual=s;

        InicializoAtributoUsada(
                ConjuntoDeAristasUsadasPorVezPrimeraEnCaminoActual,False);
        //Se inicializa a False el atributo Usada para las aristas que se intentaron utilizar por
        //primera vez.

        KParcialUsado = KParcialUsado - CostoTotalAristas(
                        ConjuntoDeAristasUsadasPorVezPrimeraEnCaminoActual);
        //Se resta a KParcialUsado el costo de las aristas que se descartaron y que
        //constituían el costo en el camino que se pretendía construir.

        ConjuntoDeAristasUsadasPorVezPrimeraEnCaminoActual={};
        //Inicializo estructura de trabajo

        ReinicializarLista(CaminoActual);
        //Inicializo estructura de trabajo

        //Se ha fallado en la construcción de un nuevo camino por lo que...

        SucesivosIntentosFallidos++;
        IncrementoSucesivosIntentosDeUsoParaCadaArista(CaminoActual);
```

```
                    SI (SucesivosIntentosFallidos >
                                ToleranciaDeSucesivosIntentosFallidos) ENTONCES
                        EsPosibleContinuar=FALSE;
                    FIN SI
                FIN SI //EncontreUnAtajo
            FIN SI //PudeCostoYLargo
        FINMIENTRAS
        FIN FUNCION ConstruyoGrafoDeCaminosFactiblesS-T
```

```
/+-----------------
                    Algoritmo de Búsqueda Local
                                                -----------------+/

FUNCION BusquedaLocalGrasp (GConstructor *I*, D *I*, K *I*, GBusquedaLocal *O*);
Parámetros:
  *I* - GConstructor:      Grafo asociado al problema original que se debe resolver.
  *I* - D:                 Parámetro del problema original que especifica la restricción para el largo de los
                           caminos s-t.
  *I* - K:                 Restricción de costo para el problema original.
*O* - GBusquedaLocal:      Grafo construido por este algoritmo.

CantidadDeIntentosDeMejorasSucesivas
                           // Parámetro que representa la cantidad de veces  que se quiere
                           // redefinir el conjunto de pares de caminos.
TopeDeFracasosSucesivos
                           //Cota para la cantidad de intentos fallidos en mejorar la solución actual
                           //que se desea tolerar durante la búsqueda.
TotalParesDeCaminos
                           //En principio sería fijo, pero puede llegar a ser una lista e valores uno por
                           //cada una de las iteraciones de CantidadDeIntentosDeMejorasSucesivas

Intentos=1;
FracasosSucesivos=0;
GBusquedaLocal=GConstructor;
ConfiabilidadRealActual=ConfiabilidadReal(GBusquedaLocal);
        // Inicializo el valor de la confiabilidad real hasta ahora alcanzada

MIENTRAS (Intentos <= CantidadDeIntentosDeMejorasSucesivas) AND
                                (FracasosSucesivos < TopeDeFracasosSucesivos) HACER
        SeleccionarConjuntoDeParesDeCaminos( GBusquedaLocal *I*,
                                TotalParesDeCaminos *I*, ConjuntoParesDeCaminos *O*);


        ListarIntercambiosPosiblesEntreParesDeCaminos(ConjuntoDeParesDeCaminos *I*,
                ListaXChangesQueMejoranConf *O*, ListaXChangesQueDanHolgura *O*);

        GBusquedaLocalTemporal=GBusquedaLocal;
        AplicarIntercambios( GBusquedaLocalTemporal *I/O*,
                ListaXChangesQueMejoranConf *I*, ListaXChangesQueDanHolgura *I*);
            // Aplico los intercambios en un grafo temporal copia del original para evaluar la efectividad de este
            // paso en la Búsqueda Local.
```

```
        ConfiabilidadRealActualTemporal=ConfiabilidadReal(GBusquedaLocalTemporal);
                // A continuación verifico que haya mejorado la confiabilidad de la solución hasta ahora encontrada
                // para intercambiarla
        SI ConfiabilidadRealActualTemporal > ConfiabilidadRealActual ENTONCES
                GBusquedaLocal=GBusquedaLocalTemporal;
                ConfiabilidadRealActual=ConfiabilidadRealActualTemporal;
                FracasosSucesivos=0;
        SINO
                FracasosSucesivos++;
        FIN SI

        // No discontinúo el ciclo de iteraciones ya que tiene un límite dado por el usuario y además
        // la elección de caminos realizada por la función SeleccionarConjuntoDeParesDeCaminos() es aleatoria
        // por lo que puede darse una mejora luego de varios fracasos

        Intentos++;
FIN MIENTRAS
```

```
FUNCION SeleccionarConjuntoDeParesDeCaminos( GConstructor *I*,
                                        TotalParesDeCaminos *I*, ConjuntoParesDeCaminos *O*);
```

Parámetros:

I - GConstructor: Grafo asociado al problema original que se debe resolver.

I - TotalParesDeCaminos: Cantidad de pares de caminos propuestos que se desea.

O -ConjuntoParesDeCaminos: Conjunto de pares de caminos que se propone con esta función.

```
ParesCaminosArmados=0;
ConjuntoCaminos:=GConstructor.ConjuntoCaminos();
        // la lista de caminos estaría ordenada por confiabilidad por parte del Constructor .

ConjuntoParesDeCaminos={};
        MIENTRAS (ParesCaminosArmados<TotalParesDeCaminos)
                        Y (Cardinal(ConjuntoCaminos) >1 ) HACER
                {c1,c2}=SeleccionarAlAzarDosCaminos(ConjuntoCaminos);

                //garantizo a continuación que un camino no este en más de un par
                ConjuntoCaminos=ConjuntoCaminos-{c1,c2};
                ConjuntoParesDeCaminos U= {(c1,c2)};

                ParesCaminosArmados++;
                        //actualizo el total de pares de caminos armados

        FIN MIENTRAS
FIN FUNCION SeleccionarConjuntoDeParesDeCaminos
```

FUNCION ListarIntercambiosPosiblesEntreParesDeCaminos(ConjuntoDeParesDeCaminos *I*, ListaXChangesQueMejoranConf *O*, ListaXChangesQueDanHolgura *O*);

Parámetros:
I - ConjuntoDeParesDeCaminos: Conjunto de pares de caminos propuestos en la fase anterior.
O - ListaXChangesQueMejoranConf: Lista de xchanges que mejoran la confiabilidad propuestos con esta función.
O - ListaXChangesQueDanHolgura: Lista de xchanges que mantienen la confiabilidad y mejoran la holgura propuestos con esta función.

//Las listas devueltas contienen xchanges factibles que se quiere proponer como posibles. Algunos de ellos pueden dejar //de ser factibles dependiendo del orden en el que se vayan aplicando.

//Dado que esta función no se encarga directamente de la aplicación de xchange, sino que solamente propone, la //evaluación de si es o no un xchange razonable se realizará sin un esfuerzo computacional exhaustivo. Dejando este //trabajo entonces para una fase posterior en dónde se hagan efectiva la aplicación de los xchanges.

```
Cabeza1=1;
        //Confiabilidad del primer tramo del Camino c1 hasta la arista en curso(exclusive)

Cabeza2=1;
        //Confiabilidad del primer tramo del Camino c2 hasta la arista en curso(exclusive)

Cola1=Confiabilidad(c1);
        //Confiabilidad del último tramo del Camino c1 desde la arista en curso(exclusive)

Cola2=Confiabilidad(c2);
        //Confiabilidad del último tramo del Camino c2 desde la arista en curso(exclusive)

RankingDeIntercambios={};
PARA CADA ParDeCaminos (c1, c2) HACER
        Cabeza1=1;
                //Confiabilidad del primer tramo del Camino c1 hasta la arista e1 sin considerarla.

        Cola1=Confiabilidad(c1);
        PARA CADA arista e1 perteneciente(c1) HACER
                Cola1=Cola1/Conf(e1);
                Cabeza2=1;
                Cola2=Confiabilidad(c2);
                PARA CADA arista e2 perteneciente(c2) HACER
                        Cola2=Cola2/Conf(e2);
                        SI NoExistenVerticesComunes(e1, e2) ENTONCES
                                SI ExistenAristasParaX-Change(e1, e2) ENTONCES
                                        Confc1'=Cabeza1*Conf(AristaNexo1)*Cola2;
                                                //AristaNexo1 vincula a Cabeza1 con Cola2.

                                        Confc2'=Cabeza2*Conf(AristaNexo2)*Cola1;
                                                //AristaNexo2 vincula a Cabeza2 con Cola1.
```

```
                                    MejoraConfiabilidad= (Conf(c1)+(1-Conf(c2))*Conf(c2) ) <
                                                (Confc1'+(1-Confc2')*Confc2' );
                                        //Estimador de Comparación de Confiabilidades propuesto por
                                        //Héctor.
                                    MantieneConfiabilidad=
                                                (Conf(c1)+(1-Conf(c2))*Conf(c2))=
                                                        (Confc1'+(1-Confc2')*Confc2');
                                    SI MejoraConfiabilidad ENTONCES
                                        IngresoXChange(ListaXChangesQueMejoranConf);
                                        // no se considera la restricción de costo pues puede
                                        // darse que sea factible ese cambio cuando se genere
                                        // alguna holgura luego de la aplicación de un
                                        //xchange o por que las nuevas aristas ya están
                                        //siendo utilizadas haciendo que el costo de dichas aristas
                                        //ya este contemplado.
                                    FIN SI
                                    SI (costo(e1)+costo(e2)> costo(cruzadas) )
                                                Y MantieneConfiabilidad ENTONCES
                                        IngresoXChange(ListaXChangesQueDanHolgura);
                                    FIN SI
                                    // De considerar lo pedido por Héctor de los pesos sobre las aristas se
                                    //puede considerar un X-change que solo mejore las distancias.
                                FIN SI
                            FIN SI
                            Cabeza2=Cabeza2*Confiabilidad(e2);
                        FIN PARA
                        Cabeza1=Cabeza1*Confiabilidad(e1);
                    FIN PARA
                FIN PARA
                FIN FUNCION ListarIntercambiosPosiblesEntreParesDeCaminos
```

```
                FUNCION AplicarIntercambios(GBusquedaLocal *I/O*,
                            ListaXChangesQueMejoranConf *I*, ListaXChangesQueDanHolgura *I*);
```

Parámetros:

I/O - GBusquedaLocal:	Grafo actualizado por esta función.
I - ListaXChangesQueMejoranConf:	Lista de xchanges que mejoran la confiabilidad propuestos en la fase anterior.
I - ListaXChangesQueDanHolgura:	Lista de xchanges que mantienen la confiabilidad y mejoran la holgura propuestos en la fase anterior.

```
                MIENTRAS No_Vacio(ListaXChangesQueMejoranConf) HACER
                    //Como me interesa maximizar la confiabilidad puedo irme sin aplicar xchanges de
                    // ListaXChangesQueDanHolgura

                    XChangeConfSobreCostoObjetivo=
                                MejorXChangeSegunConfSoreCosto(ListaXChangesQueMejoranConf);
                                // Selecciona el mejor xchange de la lista según conf. sobre costo. Podría considerar(mejora1)

                        SI HolguraEnCosto>=Costo(XChangeConfSobreCostoObjetivo) ENTONCES
                            // Costo(XChangeConfSobreCostoObjetivo) debe realizar la diferencia entre costos de las nuevas
                            //aristas y los costos de las aristas a intercambiar, teniendo en cuenta que se cumple que el costo es
                            //cero para las aristas nuevas si ya están en el grafo construido. Lo mismo sucede para las aristas a
                            //intercambiar.
```

```
        AplicarXChange(XChangeConfSobreCostoObjetivo);
        EliminarXChangesAfectados(XChangeConfSobreCostoObjetivo,
                                  ListaXChangesQueMejoranConf);
            // Elimina de la  lista los xchanges que tienen alguna arista en común con el xchange
            //XChangeConfSobreCostoObjetivo (luego de aplicar dicho  xchange se volvieron infactibles),
            //elimina también a  XChangeConfSobreCostoObjetivo.

        EliminarXChangesAfectados(XChangeConfSobreCostoObjetivo,
                                  ListaXChangesQueDanHolgura);
            // análogo anterior
    SINO // Como no tengo holgura suficientemente para aplicar el xchange que me da mejor
        // confiabilidad al mejor costo aplico xchange del ListaXChangesQueDanHolgura que me den holgura
        //suficiente
        // Se podría considerar (mejora2)
        MIENTRAS (HolguraEnCosto<Costo(XChangeConfSobreCostoObjetivo)) Y
                 (NoVacio(ListaXChangesQueDanHolgura)) ENTONCES
            XChangeHolgura=
                MejorXChangeEnDarHolgura (ListaXChangesQueDanHolgura);
                // Selecciona el mejor xchange de la lista en dar holgura
            Aplicar_xchange(XChangeHolgura);
                // actualiza la variable HolguraEnCosto

            // A continuación verifico si el xchange que mejora la holgura afecta al xchange objetivo en
            // cuyo caso lo señalizo
            SI Afecta(XChangeHolgura,XChangeConfSobreCostoObjetivo)
                                                    ENTONCES
                XChangeConfSobreCostoObjetivo =NULO;
            FIN SI
            EliminarXChangesAfectados(XChangeHolgura,
                                      ListaXChangesQueMejoranConf);
            EliminarXChangesAfectados(XChangeHolgura,
                                      ListaXChangesQueDanHolgura);
        FIN MIENTRAS

        //A continuación verifico que no haya perdido la referencia al xchange objetivo en la aplicación de los
        //xchanges que mejoraban la holgura  y que no pude alcanzar la holgura necesaria.
        SI  (XChangeConfSobreCostoObjetivo NO ES NULO)  Y
           (HolguraEnCosto<Costo(XChangeConfSobreCostoObjetivo)))  ENTONCES

            // Se cumple que ListaXChangesQueDanHolgura es vacío en este SI-ENTONCES.
            //A pesar de que no pude alcanzar la holgura necesaria para aplicar  el  xchange
            //XChangeConfSobreCostoObjetivo debo realizar una última recorrida.
            // Se realiza solo una última recorrida pues es imposible que se generen nuevas holguras por
            //como esta armada la ListaXChangesQueMejoranConf

            MIENTRAS No_Vacio(ListaXChangesQueMejoranConf) HACER
                XChangeConfSobreCostoObjetivo=
                    MejorXChangeSegunConfSoreCosto(
                                      ListaXChangesQueMejoranConf);
                    //Elige el mejor xchange de la lista según conf. sobre costo. (mejora1)
                SI HolguraEnCosto>=XChangeConfSobreCostoObjetivo.costo
                                                    ENTONCES
                    // Aplico el xchange por ser factible. Podría considerar (mejora1).
```

```
                                        AplicarXChange(XChangeConfSobreCostoObjetivo);
                                                // Se actualiza la variable HolguraEnCosto.

                                        EliminarXChangesAfectados(
                                                XChangeConfSobreCostoObjetivo,
                                                        ListaXChangesQueMejoranConf);
                                                //Elimina de la lista los xchanges que tienen alguna arista en
                                                //común con el X-change XChangeConfSobreCostoObjetivo
                                                //elimina también a XChangeConfSobreCostoObjetivo.
                                                //Considero para ello solo elListaXChangesQueMejoranConf ya
                                                //que el ListaXChangesQueDanHolgura es vacío.
                                SINO
                                        // Solo elimino el xchange XChangeConfSobreCostoObjetivo el cual no
                                        // es factible
                                        EliminarXChange (XChangeConfSobreCostoObjetivo,
                                                        ListaXChangesQueMejoranConf);
                                FIN SI
                        FIN MIENTRAS
                SINO
                        // Aplico el xchange por ser factible. Se podría considerar (mejora1).
                        AplicarXChange(XChangeConfSobreCostoObjetivo);
                        EliminarXChangesAfectados(XChangeConfSobreCostoObjetivo,
                                                        ListaXChangesQueMejoranConf);
                                //Elimina del la lista los xchanges que tienen alguna arista en común
                                //con el xchange XChangeConfSobreCostoObjetivo (luego de aplicar
                                //dicho xchange se volvieron infactibles), elimina también a
                                //XChangeConfSobreCostoObjetivo.

                        EliminarXChangesAfectados(XChangeConfSobreCostoObjetivo,
                                                        ListaXChangesQueDanHolgura);
                                //Análogo al comentario anterior.
                FIN SI
        FIN SI
FIN MIENTRAS
FIN FUNCION AplicarIntercambios;
```

5.3. Algoritmos Genéticos

5.3.1. Introducción

El algoritmo basado en la metaheurística Algoritmos Genéticos fue desarrollado reutilizando en lo posible las ideas de GRASP, ganando así la comparación más efectiva de ambas metaheurísticas, pero dificultando la elaboración de la estrategia misma.

En un principio se planteó la posibilidad de que los individuos que integrarían la población serían caminos simples de s-t, que en conjunto formaran el grafo solución que se buscaba para el problema dado.

Sin embargo, el materializar esta alternativa de orientar un Algoritmo Genético a la construcción de caminos de s a t no quedan claras las características que deben poseer los caminos en la población final. La red más confiable esta formada por caminos que en conjunto forman la mejor solución. Véase que no necesariamente todos estos caminos son los mejores o los peores. La realidad indica que la única característica que se cumple en la red más confiable, es que el conjunto de caminos es el mejor conjunto.

Esta fue una de las razones por las cuales no se implementó el hecho de que los integrantes de la población fueran caminos de s a t, al no ser fácil determinar que tipo de población de caminos se pretende alcanzar. Por otro lado, aún teniendo una población lo suficientemente diversa en la cual se disponga de los caminos necesarios para armar el mejor grafo, sería necesario llevar a cabo un procedimiento computacionalmente costoso en tiempo y recursos para obtener el mejor subconjunto de todos los posibles para ese conjunto. Notar que teniendo un conjunto finito de los mejores caminos puede parecer computacionalmente sencillo construir la solución final. Sin embargo, la identificación de todos los subconjuntos de un conjunto de caminos dado tiene una complejidad del orden de 2n, dónde n es el número total de caminos que se quiere considerar, a lo que se agrega la complejidad de evaluar cada una de las posibilidades.

La estrategia que finalmente se terminó implementando considera a los individuos como grafos. En ella se genera, primeramente, la población inicial invocando al constructor GRASP para obtener cada individuo de la misma.

Durante la evolución se determina si corresponde mutar individuos. En cuyo caso, para cada individuo a mutar se determina el tipo de mutación a realizar, generando así un porcentaje dado de individuos mutados que pasan a forma parte de una población intermedia. Se completa el número de individuos en la población intermedia llevando a cabo tantas recombinaciones/reproducciones (al azar) como sea necesario. Una vez completa, se procede a la selección de los n mejores individuos considerando en conjunto la población actual y la intermedia. Como resultado de esa selección se obtiene la población siguiente. Al completar el número de evoluciones determinadas se devuelve el mejor individuo obtenido.

La mutación puede redundar en una modificación leve o radical del individuo.
La mutación leve, denominada Mutación Débil, consiste primeramente, en recortar el individuo a mutar. Esto se logra construyendo una solución factible invocando al

Constructor de GRASP con una fracción aleatoria del presupuesto y tomando como referencia las aristas del individuo. Finalmente se enriquece el individuo así obtenido con aristas del grafo original, realizando una nueva invocación al Constructor de GRASP con el total de presupuesto y tomando como referencia precisamente el Grafo Original Simplificado.
La mutación radical, que denominamos Mutación Fuerte, realiza primeramente, el complemento del individuo a mutar respecto del Grafo Original Simplificado. Entiéndase como acepción par el término "complemento" la usual en Teoría de Grafos. Finalmente, a los efectos de garantizar que la solución no supere la restricción impuesta para el costo, se aplica el Constructor de GRASP considerando el presupuesto asociado al problema.

Conviene señalar que lo anteriormente expresado es una simplificación del algoritmo llevado a cabo a los efectos de su mejor entendimiento. No resulta adecuado a este nivel hacer una descripción con un mayor detalle; éste debe ser obtenido al referencia el pseudocódigo adjunto.

Puntualización

A lo largo de la literatura se habla de una mutación leve que permita diversificar una población existente. No existe el concepto de diversificación radical. Entendemos oportuno introducir el concepto de mutación radical, el cual consiste en hacer cambios sustanciales durante la mutación. La mutación tradicional promete una diversificación mínima de la población en un entorno de sus características. Por el contrario, la mutación radical promete una modificación mayor en la población que la haría escaparse de sus características promedio.
Otra forma de visualizarlo es considerar que la mutación tradicional resulta en un salto pequeño y conservador en el espacio de soluciones. Por el contrario, la mutación radical ofrece un salto mayor y aventurero en el espacio de soluciones.
Por esta razón se llevó a cabo la implementación de dos variantes de mutación: Mutación Débil y Mutación Fuerte, a los efectos de que fuera posible la elección de una u otra o la combinación de las mismas.

5.3.2. Pseudocódigo

```
/+++++++++++++++++++++++++++++++++++++++++++++++++++++++/
/+                                                     +/
/+              ALGORITMOS GENÉTICOS                   +/
/+                                                     +/
/+++++++++++++++++++++++++++++++++++++++++++++++++++++++/


/+-----------------
                    Algoritmo Principal
                                        -----------------+/

FUNCION GeneticAlgorithm (GOriginal *I*, GSolucionGenetico *O*);
Parámetros:
  *I* - GOriginal:          Grafo asociado al problema original que se debe resolver.
*O* - GSolucionGenetico:    Grafo construido por este algoritmo.

        ProbabilidadMutacionFuerte
                            //Probabilidad de que al ocurrir una mutación se tenga una mutación
                            //fuerte.

        PorcentajeDeGeneracionesAEsperarParaMutar
                            //Porcentaje del total de generaciones que es preciso esperar para que se produzca
                            //una posible mutación (por defecto 5%).

        PorcentajeDeIndividuosAMutar
                            //Porcentaje de individuos que sufren alguna mutación.(por defecto 1%)
                            //En la literatura "A Genetic Algorithm  Tutorial (Technical Report CS-93-103
                            //November 10, 1993) - Darrell Whitley" Pág 6.

        GeneracionesAEsperarParaMutar=TopeDeGeneraciones *
                                        PorcentajeDeGeneracionesAEsperarParaMutar;

        TotalIndividuosAMutar=PoblacionTotal * PorcentajeDeIndividuosAMutar;

        PoblacionActual=GenerarPoblacionInicial(PoblacionTotal);
                //Genero la población inicial

        NumeroDeGeneracionesLlevadasACabo=0;

        MIENTRAS SeDebaContinuar(NumeroDeGeneracionesLlevadasACabo,
                                        TopeDeGeneraciones) HACER
                SI GeneracionesAEsperarParaMutar=0 ENTONCES
                        MutacionDeIndividuos(PoblacionActual,TotalIndividuosAMutar,
                                        ProbabilidadMutacionFuerte);
                // Inicializo a continuación la espera para realizar la próxima mutación
```

```
                GeneracionesAEsperarParaMutar=TopeDeGeneraciones *
                                        PorcentajeDeGeneracionesAEsperarParaMutar;
            SINO
                GeneracionesAEsperarParaMutar - - ;
            FIN SI

            PoblacionActual=GenerarSiguientePoblacion(PoblacionActual, PoblacionTotal,
                                                    PorcentajeAMantenerDeMejores);
                // Renuevo la población actual
            NumeroDeGeneracionesLlevadasACabo++;
        FIN MIENTRAS

        MejorIndividuo=ElegirMejorIndividuo(PoblaclionActual,
                                                PorcentajeAMantenerDeMejores);
        RETORNAR MejorIndividuo;
    FIN FUNCION GeneticAlgorithm
```

```
    FUNCION ElegirMejorIndividuo(PoblacionActual *I*, PorcentajeAMantenerDeMejores *I*,
                                                        MejorIndividuo *O*);
```

Parámetros:

I - PoblacionActual:	Conjunto de individuos en la población actual.
I - PorcentajeAMantenerDeMejores:	Porcentaje que se debe mantener de los mejores individuos. Usado también para seleccionar un subconjunto utilizando un algoritmo de cálculo de confiabilidad estimativo y no exacto.
O - MejorIndividuo:	Mejor Individuo encontrado por esta función.

```
        //A continuación se elige un subconjunto de los mejores usando un estimador de la
        //confiabilidad real (MCC) para realizar el cálculo exacto (pesado) sobre este subconjunto
        //selecto y no en la población total.
        MejoresIndividuos=ElegirMejoresIndividuos(PoblacionActual,
                                                PorcentajeAMantenerDeMejores);

        //Calculando la confiabilidad exacta sobre el subconjunto selecto elijo el mejor individuo.
        MejorIndividuo=IndividuoConMejorConfiabilidadReal(MejoresIndividuos);
    FIN FUNCION ElegirMejorIndividuo
```

```
    FUNCION GenerarPoblacionInicial(PoblacionTotal *I*, PoblacionInicial *O*);
```

Parámetros:

I - PoblacionTotal:	Cantidad de individuos en la población actual.
O- PoblacionInicial:	Conjunto de individuos que integran la población actual generados por esta función.

```
        NumeroDeIndividuos=0;

        MIENTRAS NumeroDeIndividuos<PoblacionTotal HACER
            NuevoIndividuo= ConstructorGrasp(GOriginal, Diametro, Presupuesto);
            AgregoIndividuo(NuevoIndividuo, PoblacionInicial);
            NumeroDeIndividuos++;
        FIN MIENTRAS
    FIN FUNCION GenerarPoblacionInicial
```

```
FUNCION MutacionDeIndividuos(PoblacionActual *I/O*, TotalIndividuosAMutar *I*,
                                                    ProbabilidadMutacionFuerte *I*);
```

Parámetros:

I/O- PoblacionActual:	Conjunto de individuos en la población actual.
I - TotalIndividuosAMutar:	Cantidad de individuos que sufrirán una mutación.
I - ProbalidadMutacionFuerte:	Probabilidad de que al ocurrir una mutación se tenga una mutación fuerte.

```
    NumeroDeIndividuosMutados=0;
    MIENTRAS NumeroDeIndividuosMutados<TotalIndividuosAMutar HACER
        IndividuoAMutar=SorteoIndividuo(PoblacionActual);
        SorteoMutacion=Sortear(0,1);
                //Sortea un valor entre 0 y 1.

        SI SorteoMutacion <= ProbabilidadMutacionFuerte ENTONCES
            NuevoIndividuoMutado= MutacionFuerte(IndividuoAMutar);
        SINO
            NuevoIndividuoMutado= MutacionDebil(IndividuoAMutar);
        FIN SI
        ModificoPoblacion(IndividuoAMutar,NuevoIndividuoMutado, PoblacionActual);
        NumeroDeIndividuosMutados++;
    FIN MIENTRAS
FIN FUNCION MutacionDeIndividuos
```

```
FUNCION GenerarSiguientePoblacion(PoblacionActual *I*, PoblacionTotal *I*,
                                PorcentajeAMantenerDeMejores *I*, PoblacionSiguiente *O*);
```

Parámetros:

I - PoblacionActual:	Conjunto de individuos en la población actual.
I - PoblacionTotal:	Cantidad de individuos en la población actual.
I - PorcentajeAMantenerDeMejores:	Porcentaje que se debe mantener de los mejores individuos. Usado también para seleccionar un subconjunto utilizando un algoritmo de cálculo de confiabilidad estimativo y no exacto.
O- PoblacionSiguiente:	Conjunto de individuos que constituirán la población siguiente generados por esta función.

```
    PoblacionSiguiente={};
    MejoresIndividuosPoblacionActual=ElegirMejoresIndividuos(PoblacionActual,
                                                    PorcentajeAMantenerDeMejores);
    AgregoIndividuos(MejoresIndividuosPoblacionActual, PoblacionSiguiente);

    NumeroDeIndividuosGenerados=PorcentajeAMantenerDeMejores* PoblacionTotal;
        //Actualizo el valor de total de individuos generados considerando el porcentaje que se mantuvo de la
        //población actual.

    MIENTRAS NumeroDeIndividuosGenerados< PoblacionTotal HACER
        IndividuoAReproducir1=SorteoIndividuo(PoblacionActual);
        IndividuoAReproducir2=SorteoIndividuo(PoblacionActual);
        ReproducionSexuada(IndividuoAReproducir1,IndividuoAReproducir2,
                                                    NuevoIndividuo);
```

```
        AgregoIndividuo(NuevoIndividuo, PoblacionSiguiente);
        NumeroDeIndividuosGenerados++;
    FIN MIENTRAS
FIN FUNCION GenerarSiguientePoblacion
```

```
FUNCION ElegirMejoresIndividuos(  PoblacionActual *I*,
                                  PorcentajeAMantenerDeMejores *I*, MejoresIndividuos *O*);
```

Parámetros:

I - PoblacionActual:	Conjunto de individuos en la población actual.
I - PorcentajeAMantenerDeMejores:	Porcentaje que se debe mantener de los mejores individuos. Usado también para seleccionar un subconjunto utilizando un algoritmo de cálculo de confiabilidad estimativo y no exacto.
O - MejoresIndividuos:	Conjunto de los mejopres individuos de la población actual. Su tamaño en porcentaje respecto de la población actual esta dado por PorcentajeAMantenerDeMejores.

```
    //Esta función elige un porcentaje de los mejores individuos sin calcular la
    //confiabilidad real, solamente considera el estimador basado en las confiabilidades de los caminos
FIN FUNCION ElegirMejoresIndividuos
```

```
/+------------------
```

Algoritmo de la Mutación

```
------------------+/
```

```
FUNCION MutacionFuerte (IndividuoAMutar *I*,  IndividuoMutado *O*);
```
Parámetros:

I - IndividuoAMutar:	Individuo que se sufrirá una mutación fuerte.
O - IndividuoMutado:	Individuo que se sufriera una mutación fuerte.

```
        MutadoGInicial=Complemento (GInicial, GOriginal);
                // Complemento de Ginicial según GOriginal
        MutadoGInicial=Simplifacion(MutadoGInicial);
        IndividuoMutado =ConstructorGrasp(MutadoGInicial)
        RETORNAR (IndividuoMutado) ;
FIN FUNCION
```

```
FUNCION MutacionDebil (IndividuoAMutar *I*,  IndividuoMutado *O*);
```
Parámetros:

I - IndividuoAMutar:	Individuo que se sufrirá una mutación débil.
O - IndividuoMutado:	Individuo que se sufriera una mutación débil.

```
        IndividuoMutado =BusquedaLocalGrasp(IndividuoAMutar);
        RETORNAR (IndividuoMutado) ;
FIN FUNCION
```

```
/+------------------
```

Algoritmo de Reproducción

```
------------------+/
```

```
FUNCION ReproduccionSexuada (GPadre1 *I*, GPadre2 *I*, GHijo *O*);
```
Parámetros:

I - Gpadre1:	Grafo que constituirá uno de los padres en la reproducción sexuada.
I - Gpadre2:	Grafo que constituirá otro de los padres en la reproducción sexuada.
O - GHijo:	Grafo resultado de la reproducción sexuada llevada a cabo.

```
        ListaDeCaminos={};
        SeleccionoCaminos(GPadre1, CaminosSeleccionados);
                // CaminosSeleccionados contiene los caminos que se seleccionan del grafo Gpadre1.

        AgregoCaminos(ListaDeCaminos, CaminosSeleccionados);
                // ListaDeCaminos contendrá todos los caminos seleccionados de los grafos Padres.

        SeleccionoCaminos(GPadre2, CaminosSeleccionados);
                // CaminosSeleccionados contiene los caminos que se seleccionan del grafo Gpadre2.
```

```
        AgregoCaminos(ListaDeCaminos, CaminosSeleccionados);

        GTemporal=GrafoGeneradoPorCaminos(ListaDeCaminos);
                        //Esta función devuelve el grafo generado con los caminos de ListaDeCaminos

        GTemporalPrima=Simplificacion(GrafoTemporal);
        GHijo=ConstructorGrasp(GTemporalPrima);

        //Otra variante para AlgoritmosGenéticos que no sería válida es
                //          GHijo=BusquedaLocalGrasp(GTemporalPrima);
                //pues no la búsqueda local no resuelve el caso de no factibilidad que ya  cumple
                //GTemporalPrima. Se puede ver con un poco de esfuerzo que no brindaría aporte alguno, el
                //aplicar previamente un procedimiento que factibilizara a GTemporalPrima.

        //Y no vale aplicar luego una BusquedaLocalGrasp, la cual sería relevante a nuestros intereses, por ser una
        //variante de  AlgoritmosGenéticos con GRASP, la que sería ...
                //GHijo=Grasp(GTemporalPrima, UnicaIteracion);
        //Esta variación importante se llevará a cabo en etapas finales la cual constituirá una alternativa aceptable de
        //combinar GRASP y Algoritmos Genéticos
FIN FUNCION
```

```
/+-----------------

                        Algoritmo de Selección

                                -----------------+/
```

```
// En primera instancia consideramos el algoritmo de selección más sencillo consistente en :
        // En lo que respecta a seleccionar los candidatos para la reproducción: los mismos serán
                // determinados aleatoriamente, argumentando que la población actual es mejor o
                //  igual a la anterior.
        // En lo que respecta seleccionar la nueva población: la nueva población será la población de
                // individuos obtenida al aplicar los algoritmos de reproducción a la población actual.

// Esta elección de criterios de selección tendría un respaldo fuerte si se pudiera argumentar que la solución obtenida luego
// de aplicar los criterios de mutación  y de reproducción  se obtienen individuos que no son peores que sus padres (tener
// cuidado con la definición de un algoritmo que mantenga en el peor caso la calidad del padre para la mutación ya que
// puede corromper la esencia de la mutación).

// En el caso en que no se pudiera llegar a afirmar lo antedicho, se puede mantener incambiado el primer criterio (el de
// selección de los candidatos) pero se debe modificar el segundo pues se puede perder la solución globalmente óptima en
//alguna etapa de la evolución.
```

5.4. Algunos Detalles de Implementación

Si se pretende utilizar o modificar el código fuente desarrollado, recomendamos fuertemente la lectura de esta sección, en la que incluimos una descripción resumida y en alto nivel de lo implementado.

Hemos hecho importantes esfuerzos para describir los aspectos de bajo nivel en forma clara y entendible, teniendo siempre presente el evitar incluir en esta sección trozos del código implementado, ya que dificultaría y oscurecería su contenido. Por tal razón no debe perderse de vista que lo puntualizado en esta sección no resultó tan simple en la práctica, ya que deliberadamente estamos omitiendo aspectos de bajo nivel, los que resultan inevitables en la fase de implementación y que en suma terminan constituyendo la mayor parte del costo de desarrollo asociado a la misma. Debe tenerse presente entonces, que el total de lo presentado aquí es un resumen de lo que puede encontrarse al leer el código fuente implementado.

La primer tarea vinculada a la implementación que llevamos a cabo constó en identificar los objetos básicos con los que deberían operar nuestros algoritmos. Debido a que durante dicha actividad se fueron identificando objetos compuestos llevamos a cabo un refinamiento del análisis identificando los objetos básicos que los componían. El resultado de este proceso se ve reflejado en las clases C++ que diseñamos e implementamos, clases que presentaremos a continuación en esta sección.

El primer objeto básico identificado es el objeto “grafo”. El tipo grafo necesario para nuestros algoritmos debería contener y manejar ciertas estructuras de datos que en primera instancia no se incluirían en una implementación usual de grafos, tal es el caso de la estructura que almacena los caminos explícitos que en conjunto conforman el grafo en cuestión. Por este motivo, desarrollamos una clase grafo básica, la que provee todas las operaciones usuales sobre grafos simples no dirigidos cuya estructura de datos es la necesaria para soportar estas operaciones, clase que puede ser perfectamente reutilizada y ampliada en trabajos futuros de acuerdo a las necesidades que se identifiquen.

Luego, a partir de la misma y utilizando el mecanismo de herencia de C++, implementamos clases para representar grafos tal como lo necesitan nuestros algoritmos. Así, logramos simultáneamente dos objetivos, por un lado una implementación para objetos del tipo “grafo” suficientemente genérica que puede fácilmente reutilizarse en trabajos futuros, y por otro la extensión a dicha implementación necesaria para este trabajo (tanto en funcionalidades como en eficiencia).

A continuación presentamos información relativa a los diferentes objetos tipo grafo implementados, así como una indicación de las relaciones existentes entre ellos.

Clase C++	Resumen descriptivo	Hereda de la clase ...
GrafoKDiamConf	Implementa el tipo grafo no dirigido. Provee las operaciones usuales sobre grafos, además de las necesarias para el manejo de archivos de grafos exportados por HEIDI. Encapsula una estructura eficiente tanto en el uso de memoria como en el tiempo de ejecución de los métodos que exporta.	
GrafoKDiamConfAmpliado	Amplía la clase GrafoKDiamConf, proporcionando ciertos métodos necesarios en nuestros algoritmos. Encapsula además estructuras de datos que éstos necesitan.	GrafoKDiamConf
GrafoKDiamConfGRASP	Complementa lo proporcionado por la clase GrafoKDiamConfAmpliado con las estructuras de datos faltantes para la implementación del algoritmo GRASP y proporciona todos los métodos necesarios para la ejecución del mismo.	GrafoKDiamConfAmpliado
GrafoKDiamConfGA	Utiliza la misma estructura de datos implementada en su clase base, pero exporta solamente métodos relacionados con el algoritmo genético que encapsula. Reutiliza código ya implementado en su clase base. Esto es así pues el algoritmo GA se desarrolló sobre la base de ideas generadas para el algoritmo GRASP.	GrafoKDiamConfGRASP

Además de los objetos del tipo “grafo” se hizo necesario el representar otros objetos, tanto para utilizarlos en la implementación de las estructuras de datos propias de los grafos, como en las estructuras de datos auxiliares que se utilizarían en los algoritmos a implementar. Estos objetos se implementaron por medio de clases a las que dimos en llamar “clases básicas”.

Estas “clases básicas” proporcionan las abstracciones que representan los objetos compuestos más elementales y básicos de trabajo (como ser caminos, listas, conjuntos), ocultando los aspectos oscuros de implementación de los mismos. Posteriormente a su implementación y a su uso pudimos comprobar en la práctica que facilitaron enormemente la tarea de identificar las fuentes de “Segmentation Faults”, problema común de difícil identificación y corrección en códigos C++, en particular en códigos complejos e intrincados. Por otro lado facilitan enormemente las tareas de mantenimiento que puedan llegar a ser necesarias en futuros trabajos que reutilicen lo hecho durante este taller.

Estas “clases básicas” y sus interrelaciones se encuentran resumidas en la siguiente tabla:

Clase básica	Resumen descriptivo	Clase utilizada
ArrayGenericoFlexible	Implementa el tipo de datos array genérico, es decir, implementa un tipo de datos que puede instanciarce en arrays de cualquier tipo. El sufijo “flexible” se debe a que la estructura de datos que encapsula cambia su tamaño dinámicamente (en tiempo de ejecución) para ajustarse a las necesidades del algoritmo que utiliza objetos de este tipo, proporcionando de esta manera lo equivalente a disponer de arrays de tamaño infinito. Simultáneamente proporciona un tiempo de acceso a los datos de orden O(1).	
ConjuntoGenerico	Implementa un tipo genérico para representar conjuntos, por lo que permite instanciar objetos cuyo tipo sea “conjunto de T”, siendo T un tipo cualquiera. Por ejemplo permite definir trivialmente el tipo ConjuntoDeNodos utilizado en nuestros algoritmos. Su implementación se hace sencilla utilizando la “clase básica” ListaGenerica.	ListaGenerica
ListaGenerica	Implementa el tipo de datos lista genérica, es decir, implementa un tipo de datos que puede instanciarce en listas de cualquier tipo. Las operaciones implementadas fueron básicamente las necesarias por nuestros algoritmos, no obstante se incluyeron otras para que los objetos de este tipo dispongan de una interfaz completa.	ArrayGenericoFlexible

ParGenerico	Implementa el tipo de datos par genérico, es decir, implementa un tipo de datos que puede instanciarce en pares de elementos de cualquier tipo. Útil para poder almacenar datos y datos asociados a estos en la estructura que se necesite, sin la necesidad de implementar estructuras auxiliares que mantengan la asociación entre los datos almacenados y sus datos asociados.	
ParIzquierdoGenerico	De forma análoga a la clase ParGenerico facilita la manipulación de datos que tienen asociados datos del mismo o de otro tipo. Su única diferencia con ParGenerico es que proporciona operadores relacionales para poder almacenar ordenadamente elementos de un tipo y simultáneamente su dato asociado. La denominación "Izquierdo" se debe a que estos operadores relacionales operan solamente sobre el primer elemento del par. Ej. Permite trivialmente implementar listas ordenadas de enteros, donde cada entero tiene asociado un objeto del tipo que se quiera, sin la necesidad de implementar una estructura auxiliar para almacenar la asociación.	ParGenerico

Luego de implementar estas clases básicas implementamos otras, que desde el punto de vista de los algoritmos que debíamos desarrollar resultan elementales, pero que desde el punto de vista de la implementación general en su mayoría resultan ser compuestas. Estas clases se presentan a continuación.

Clase	Resumen descriptivo	Clase utilizada
AristaKDiamConf	Implementa el tipo de datos arista, tipo de objeto frecuentemente utilizado en nuestros algoritmos.	
Camino	Implementa el tipo de datos camino, el cual representa un camino de aristas. Es utilizada para representar los caminos que son construidos por nuestros algoritmos en su búsqueda de la solución. Su implementación se hace sencilla viendo un camino como una lista de aristas.	AristaKDiamConf ListaGenerica
IndividuoPoblacionPorMaxInt	Implementa el tipo que representa individuos de una población, siendo utilizado en el Algoritmo Genético. El sufijo "PorMaxInt" indica que se ha establecido una relación de orden entre los individuos de este tipo. Consideramos que un individuo es mayor a otro si el extremo máximo de su intervalo de confianza para la Diámetro-Confiabilidad es mayor al correspondiente para el segundo individuo.	GrafoKDiamConfGA

Adicionalmente a las clases mencionadas, debimos implementar ciertas funciones que por su naturaleza debían pertenecer a una biblioteca de funciones. Las más relevantes son las siguientes:

- SortearValorAleatorio: Función que devuelve como resultado un número aleatorio en el intervalo [0,1].
- IntervalosSolapados: Función booleana que permite determinar si dos intervalos de confianza están solapados.
- IntervalosDisjuntos: Función booleana que permite determinar si dos intervalos de confianza son disjuntos.
- ConfMenorEnMCC: Función booleana que permite determinar, dados un valor de Diámetro-Confiabilidad, el número de replicaciones utilizado por el método Monte Carlo Crudo para calcularlo, y un intervalo de confianza; si el intervalo de confianza asociado a la Diámetro-Confiabilidad es menor que el especificado por parámetro.
- ConfmayorEnMCC: Análoga a la anterior, pero para el caso "mayor" y no "menor".

- ConfIgualEnMCC: Análoga a la anterior, pero para el caso “equivalente” y no “mayor”.
- Varianza: Función que devuelve la varianza del método Monte Carlo Crudo dados la Diámetro-Confiabilidad devuelto por éste y el número de replicaciones utilizado.
- RoundInt: Función que permite redondear valores “double” a valores “int”. Implementada para que nuestros códigos fueran portables entre compiladores de entornos Unix y Windows.

A continuación ampliamos el resumen descriptivo de las clases más relevantes y simultáneamente proporcionamos los detalles más destacables acerca de la implementación de las mismas:

Clase: GrafoKDiamConf

Las estructuras de datos encapsuladas por esta clase están implementadas utilizando las “clases básicas” *ListaGenerica* y *ArrayGenericoFlexible* y pueden resumirse así:

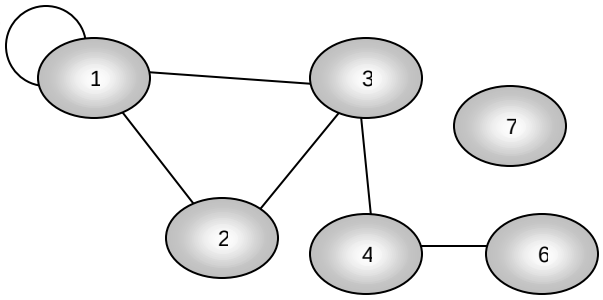
- Una matriz de adyacencia, de la que se almacena solamente la mitad superior. El contenido de la celda [i, j] de esta matriz es del tipo *int* y puede verse como un puntero a la arista entre los nodos i y j, arista almacenada en la estructura de datos para las aristas.
- Una lista de aristas, utilizada como la estructura de datos que contiene todas las aristas del grafo.
- Un array de enteros, donde en la celda i-ésima se almacena el grado del nodo i.
- Un campo del tipo *double* que contiene el costo total del grafo.

La matriz de adyacencia se implementa utilizando un campo del tipo *ArrayGenericoFlexible* instanciado con el tipo de datos **ArrayGenerico<int>.* De esta manera obtuvimos las siguientes ventajas:

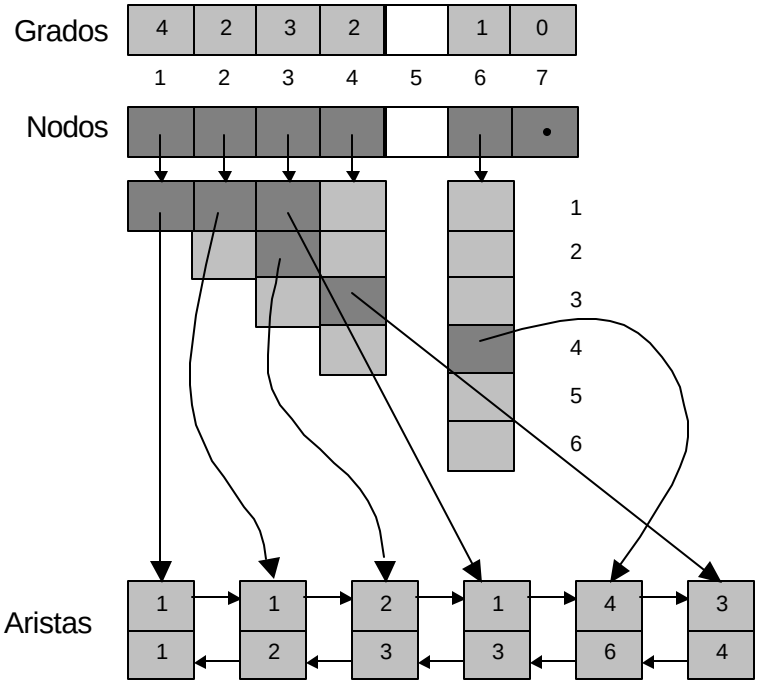
- No es obligatorio almacenar el espacio para toda la mitad superior de la matriz, sino que solamente utilizamos el espacio necesario. Por ejemplo, si en el grafo los nodos no están numerados en forma consecutiva ahorramos el espacio necesario para las columnas correspondientes a los nodos que no pertenecen al grafo.
- Al utilizar la clase *ArrayGenericoFlexible* como clase báse de la estructura de datos obtuvimos una estructura flexible, que en tiempo de ejecución adapta su tamaño en función de los nodos que son agregados o borrados.
- Se logra una velocidad de acceso a los datos cuyo orden es O(1), equivalente a la obtenida con estructuras tradicionales para almacenar matrices.

A continuación mostramos un grafo de ejemplo y la estructura básica que lo representa.

Grafo ejemplo:



Esquema simplificado de la estructura de datos utilizadas para representarlo:



Nota: Para mantener la claridad en el esquema se representa a las aristas como estructuras de datos que únicamente almacenan los nodos extremos que las definen, en el código implementado por nosotros cada arista almacena además información referente a su costo, confiabilidad y largo. Además no se representan todos los detalles de la implementación real, por ejemplo no se muestra la estructura necesaria para almacenar los iteradores definidos sobre la lista de aristas, y los punteros al elemento siguiente y al anterior no se representan como "campos" de la estructura de datos de cada elemento de esta lista.

Obsérvese que con esta estructura de datos es posible evitar almacenar siempre en forma completa la mitad superior de la matriz. En caso de que el grafo sea disperso en nodos obtenemos un importante ahorro en la memoria total ocupada al almacenar la información asociada al mismo(el espacio requerido para almacenar toda la mitad superior es del orden de n^2, siendo n el número de nodos del grafo), ahorro que se hace mayor cuanto mayor sea el grafo.

Para optimizar aún más el uso de espacio, los datos referentes a las aristas se encuentran almacenados en una lista, por lo que la única información almacenada en cada celda de la matriz es un puntero a un elemento de dicha lista, de esta manera si el grafo es disperso en aristas disminuimos aún más el consumo de espacio.

Esta estructura permite acceder a la información referente a una arista y a los grados de los nodos en O(1), y por otra parte, dado que la lista de arista es una lista doblemente encadenada, el costo en tiempo de ejecución de las operaciones "insertar arista" y "devolver arista" también es O(1).

Clase: GrafoKDiamConfAmpliado

Esta clase amplia lo proporcionado por la clase GrafoKDiamConf, y es una clase heredada de esta última; extendiendo la definición de tipos y algoritmos.

En su sección pública se encuentran (además de los constructores y operadores necesarios) dos algoritmos extremadamente útiles para el código que desarrollamos: el algoritmo de *Simplificación* y el algoritmo de *Dijkstra*.

En su sección privada, y disponibles para todas sus clases derivadas, se encuentran: el tipo de datos *exchange,* que permite representar los exchanges con los que trabajan nuestros algoritmos; y un algoritmo que permite determinar si dos nodos son conectables entre si por caminos que no excedan un determinado diámetro, este último utilizado en la implementación del cálculo de la Diámetro-Confiabilidad.

En el caso de que en futuros trabajos se desee ampliar lo proporcionado por las clases GrafoKDiamConf y GrafoKDiamConfAmpliado sin agregar más clases derivadas esta clase es la clase ideal para contener las nuevas extensiones.

Clase: GrafoKDiamConfGRASP

En la descripción de esta clase se hace necesario manejar el concepto "mejor intervalo de confianza" asociado a un conjunto de intervalos de confianza, por tal razón lo definimos a continuación.

Definición: Mejor intervalo de confianza asociado a un conjunto de intervalos de confianza.
Dado un conjunto A, cuyos elementos son intervalos de confianza, se define como mejor intervalo de confianza asociado al conjunto A, al intervalo I que cumple la siguiente condición.

$$\text{ExtremoDerecho}(I) > \max_{J \in A - \{I\}} \{ \text{ExtremoDerecho}(J) \}$$

Donde ExtremoDerecho(I) se define de la siguiente manera:

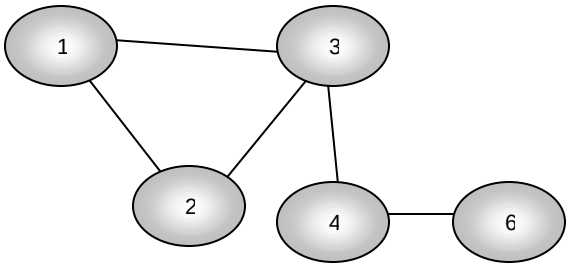
Definición: ExtremoDerecho(I)
Dado el intervalo de confianza I=[a, b], b es ExtremoDerecho(I).

Dado que el algoritmo GRASP desarrollado por nosotros, así como el algoritmo GA, necesitan la Diámetro-Confiabilidad asociada a cada solución encontrada, (básicamente para decidir si vale la pena almacenar la solución o para determinar cual es la mejor solución entre dos soluciones previamente encontradas), se hizo necesario calcularla. Por tal razón esta clase exporta en su parte pública una implementación del método Monte Carlo Crudo, con la que es posible estimar la Diámetro-Confiabilidad de una red dada.

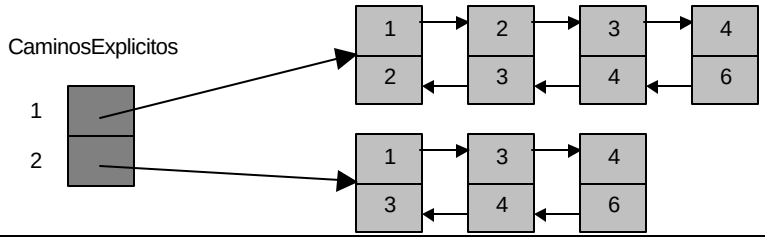
Esta clase *GrafoKDiamConfGRASP* implementa todo lo necesario para el algoritmo GRASP, hereda de la clase *GrafoKDiamConfAmpliado*, y enriquece tanto las estructuras de datos como los tipos y los métodos definidos en esta última. En lo que refiere a las estructuras de datos que proporciona destacamos la utilizada para almacenar los caminos explícitos armados por el constructor GRASP, caminos que en conjunto definen el grafo en cuestión. En la definición de esta estructura utilizamos la "clase básica" *ArrayGenericoFlexible* para definir un array flexible de punteros a caminos donde, como ya hemos dicho anteriormente, cada camino esta representado por la clase *Camino*, clase que a su vez basa su implementación en la clase *ListaGenerica*.

Gráficamente esta estructura adicional para almacenar los caminos explícitos puede visualizarse de la siguiente manera:

Grafo ejemplo:



Suponiendo que s=1 y que t=6, y además que los caminos explícitos definidos son C1=1,2,3,4,6 y C2=1,3,4,6, la estructura para almacenar estos caminos explícitos puede verse de la siguiente manera:



Nota: Por claridad en el esquema se representa a las aristas como estructuras de datos que únicamente almacenan los nodos extremos que las definen, en el código implementado por nosotros cada arista almacena además información referente a su costo, confiabilidad y largo. Además no se representan todos los detalles de la implementación real, por ejemplo no se muestra la estructura necesaria para almacenar los iteradores definidos sobre la lista de aristas, y los punteros al elemento siguiente y al anterior no se representan como "campos" de la estructura de datos de cada elemento de esta lista.

Dado que la estructura base para almacenar los punteros a los caminos es la proporcionada por la clase *ArrayGenericoFlexible* obtenemos, una estructura en la que el espacio necesario para almacenar estos punteros se ajusta dinámicamente en tiempo de ejecución según estos sean agregados o quitados, y que simultáneamente proporciona un tiempo de acceso de orden O(1) a cada uno de los caminos.

En lo que refiere a los tipos de datos, esta clase *GrafoKDiamConfGRASP* provee las definiciones de tipos necesarias para representar entre otras; la estructura de datos que almacena las aristas que se considerarán en la construcción de la solución, la estructura de datos que almacena la asociación entre estas aristas y los datos que son utilizados en nuestro algoritmo GRASP (como ser el número de veces que la arista ha sido utilizada, la lista de caminos explícitos en los que aparece la arista), pares de caminos, listas de pares de caminos y los mejores grafos solución encontrados.

Entre todas estas estructuras se destaca la utilizada para almacenar las mejores soluciones encontradas por el algoritmo GRASP. Debido a que utilizamos el método Monte Carlo Crudo para calcular la Diámetro-Confiabilidad de las redes construidas, debemos tener en cuenta el intervalo de confianza asociado a cada solución al momento de comparar dos redes para determinar cual de ellas es la más Diámetro-Confiable. Como resultado de la comparación entre dos redes no siempre se puede concluir que una sea más Diámetro-Confiable que la otra. Por tal razón tomamos la decisión de almacenar todas las soluciones que tienen posibilidades de ser la mejor solución encontrada hasta el momento.

La estructura de datos utilizada para almacenar las mejores soluciones encontradas esta basada en la clase *ArrayGenericoFlexible* (si bien la estructura construida de esta manera tiene el potencial de ajustar su tamaño conforme se encuentren soluciones, utilizamos un parámetro manejable por el usuario para acotar el número máximo de soluciones encontradas). Esta estructura para las mejores soluciones registra además el mejor intervalo de confianza entre los intervalos de todas las soluciones almacenadas.

Inicialmente esta estructura de mejores grafos se encuentra vacía. A medida que se van generando soluciones para el problema, estas se van almacenando en la misma. Este almacenamiento lo hacemos de una forma especial, garantizando que en todo momento la misma almacene solamente las soluciones candidatas a ser la mejor solución.

Dado que además del resultado devuelto por el algoritmo GRASP necesitamos indicadores del desempeño del propio algoritmo, hemos incluido en el código implementado lógica adicional para llevar a cabo un seguimiento de este desempeño. Este seguimiento consta básicamente en determinar cuando es encontrada una solución que mejora la mejor solución encontrada hasta el momento, y almacenar en dicho instante la siguiente información:

- Tiempo de procesador consumido por el algoritmo.
- Solución encontrada, que resulta ser la mejor hasta el momento.
- Iteración actual, que se corresponde con la iteración en la que se ha encontrado la mejor solución.

Este seguimiento se realiza conjuntamente con la ejecución del algoritmo GRASP y en determinados momentos la información recabada es almacenada creando un histórico resumido de la ejecución del algoritmo. Para determinar el momento en el que se debe registrar información en este histórico se utiliza el parámetro del algoritmo GRASP que indica el número de iteraciones que este debe llevar a cabo, parámetro que llamamos *Iteraciones*. El registro en el histórico se lleva a cabo finalizadas las siguientes iteraciones:

- Iteración número Iteraciones/2
- Iteración número Iteraciones*2/3
- Iteración número Iteraciones

En la implementación del repositorio de información histórica se utiliza una estructura de datos que permite, para cada uno de los momentos indicados anteriormente, almacenar la siguiente información:

- Iteración de muestreo, que se corresponde con la iteración luego de la cual se almacenó la información histórica.
- Tiempo de muestreo, que especifica el tiempo de procesador consumido por el algoritmo desde su comienzo hasta finalizar la Iteración de muestreo.
- Iteración encontrada mejor solución, que se corresponde con la iteración en la que el algoritmo encontró la mejor solución, teniendo en cuenta su ejecución hasta la Iteración de muestreo.
- Tiempo encontrada mejor solución, que se corresponde con el tiempo de procesador empleado por el algoritmo GRASP desde su comienzo hasta completar la Iteración encontrada mejor solución.
- Grafo solución, que almacena la mejor solución encontrada
- Confiabilidad solución, que almacena la Diámetro-Confiabilidad de la mejor solución.
- Número Iteraciones MCC, que almacena el número de replicaciones utilizadas por el método Monte Carlo Crudo en el cálculo de la Diámetro-Confiabilidad de la mejor solución.

Al finalizar el algoritmo GRASP, se devuelven como información estadística adicional: todas las soluciones encontradas que tienen chance de resultar ser la mejor solución para el problema, el número de soluciones devueltas, el código-resultado que indica si el usuario ha cometido un error en algún parámetro (en este caso se dispone además del número de parámetro equivocado) y si se hace necesario, el aviso de que es preciso aumentar el número de replicaciones para el cálculo de la Diámetro-Confiabilidad.

En lo que refiere a los algoritmos proporcionados por la clase *GrafoKDiamConfGRASP*, en su sección pública se destacan, además del algoritmo que implementa nuestro algoritmo GRASP y el algoritmo que implementa el cálculo de la diámetro confiabilidad utilizando el método Monte Carlo Crudo, los constructores y operadores necesarios para trabajar cómodamente con objetos del tipo *GrafoKDiamConfGRASP*, como ser el constructor copia, el constructor que permite definir objetos del tipo *GrafoKDiamConfGRASP* a partir de archivos HEIDI, constructores de conversión de tipos que permiten definir este tipo de grafos a partir de grafos del tipo *GrafoKDiamConf* y *GrafoKDiamConfAmpliado* y operadores que permiten llevar a cabo uniones y diferencias entre objetos pertenecientes a esta clase.

En su sección privada se definen todos los métodos necesarios para una implementación adecuadamente estructurada de los algoritmos que constituyen la implementación del algoritmo GRASP.

Clase: GrafoKDiamConfGA

Encapsula todo lo necesario para la implementación de nuestro algoritmo GA. Se implementó como una clase que hereda de la clase *GrafoKDiamConfGRASP* ya que, como se sabe, el algoritmo GA se basa fuertemente en el algoritmo GRASP.

Por tal razón muchas de las estructuras de datos utilizadas en esta clase son las definidas para el algoritmo GRASP en la clase *GrafoKDiamConfGRASP*; como por ejemplo la estructura que almacena las aristas que el constructor GRASP debe tener en cuenta al construir una solución. Los tipos para estas estructuras no son redefinidos nuevamente en la clase *GrafoKDiamConfGA,* sino que son reutilizadas las definiciones que se encuentran presentes en su clase base.

Dada la naturaleza de la clase *GrafoKDiamConfGA* definimos en la misma las estructuras de datos propias para el algoritmo que ésta encapsula, las más relevantes son las dos siguientes:

- Población De Individuos: Permite representar una población de individuos, donde cada nuevo individuo representa una solución factible para el problema. Esta estructura de datos fue implementada utilizando la clase ArrayGenericoFlexible, ya que una población de individuos la vemos básicamente como un array de individuos.
- Lista Ordenada de Inividuos: Es una lista de individuos, ordenada en forma decreciente según el extremo derecho del intervalo de confianza asociado a cada individuo. Para su implementación instanciamos la clase ListaGenerica con el tipo de datos que representa los individuos manejados.

Nota: Los individuos son representados por la clase IndividuoPoblacionPorMaxInt.

De forma análoga a lo que sucede con el algoritmo GRASP, fue necesario llevar a cabo un seguimiento de la evolución del algoritmo genético en su búsqueda de la mejor solución. Utilizamos la misma estrategia y una estructura de datos análoga a la que ya comentamos para la clase *GrafoKDiamConfGRASP*.

El seguimiento consta básicamente en determinar cuando el algoritmo GA encuentra una solución que mejora la mejor solución encontrada hasta el momento y almacenar en dicho instante la siguiente información:

- Tiempo de procesador consumido por el algoritmo.
- Solución encontrada, que resulta ser la mejor hasta el momento.
- Generación actual, que se corresponde con la generación en la que se ha encontrado la mejor solución.

Al igual que con el algoritmo GRASP este seguimiento se va haciendo conjuntamente con la ejecución del algoritmo GA y en determinados momentos la información recabada es almacenada creando un histórico resumido de la ejecución del este. Para determinar el momento en el que se debe registrar información en este histórico se utiliza el parámetro del algoritmo que indica el número de generaciones que este debe llevar a cabo, parámetro que en esta sección llamamos *NroGeneraciones*. El registro en el histórico se lleva a cabo finalizadas las siguientes generaciones:

- Generación número NroGeneraciones /2
- Generación número NroGeneraciones *2/3
- Generación número NroGeneraciones

En la implementación del repositorio de información histórica se utiliza una estructura de datos, que al igual que con el algoritmo GRASP permite, para cada uno de los momentos indicados anteriormente, almacenar la siguiente información:

- Generación de muestreo, que se corresponde con la generación luego de la cual se almacenó la información histórica.
- Tiempo de muestreo, que especifica el tiempo de procesador consumido por el algoritmo desde su comienzo hasta finalizar al Generación de muestreo.
- Generación encontrada mejor solución, que se corresponde con la generación en la que el algoritmo encontró la mejor solución, teniendo en cuenta su ejecución hasta la Generación de muestreo.
- Tiempo encontrada mejor solución, que se corresponde con el tiempo de procesador empleado por el algoritmo GA desde su comienzo hasta completar la Generación encontrada mejor solución.
- Grafo solución, que almacena la mejor solución encontrada.
- Confiabilidad solución, que almacena la Diámetro-Confiabilidad de la mejor solución.
- Número Iteraciones MCC, que almacena el número de replicaciones utilizadas por el método Monte Carlo Crudo en el cálculo de la Diámetro-Confiabilidad de la mejor solución.

Finalizado el algoritmo se almacenan, como información estadística adicional, todas las soluciones presentes en la última generación, ya que esta generación contiene las mejores soluciones encontradas.

De lo exportado en su sección pública destacamos el algoritmo GA que desarrollamos, los operadores de conversión de tipos que permiten definir objetos del tipo *GrafoKDiamConfGA* a partir de objetos del tipo *GrafoKDiamConf*, *GrafoKDiamConfAmpliado* y *GrafoKDiamConfGRASP*, y los diferentes constructores que permiten crear objetos pertenecientes a esta clase a partir de archivos de grafos HEIDI y a partir de objetos de grafo pertenecientes a las demás clases de grafo definidas.

5.5. Integración con HEIDI

La integración con HEIDI esta basada en el intercambio de archivos de grafos, haciendo uso de sus funcionalidades de importación y exportación de datos.

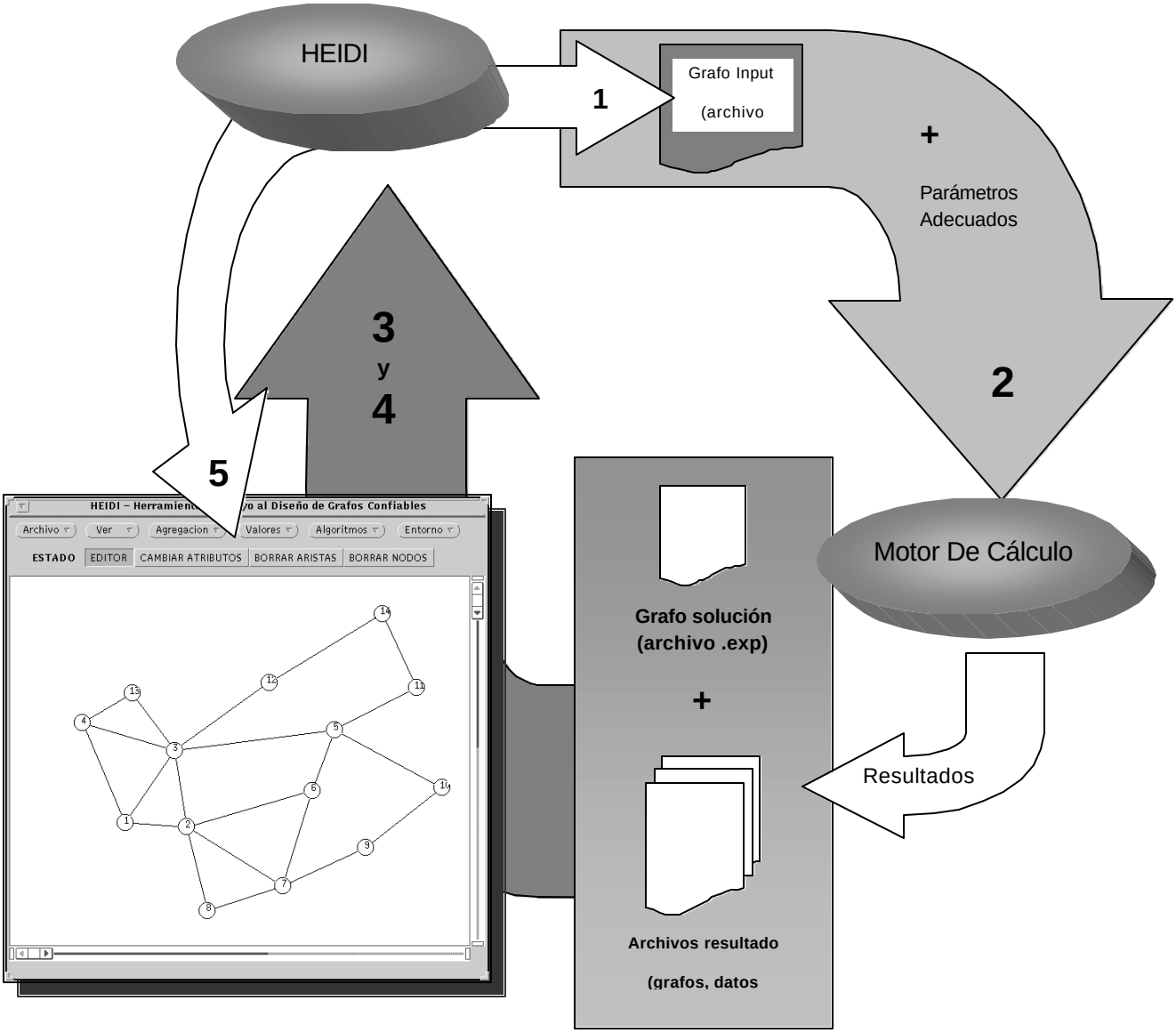
Este método tiene las siguientes características destacables:

- Permite que los algoritmos desarrollados en este taller no queden atados a la implementación de HEIDI y que puedan ser utilizados en cualquier otro ambiente de trabajo.
- Permite una integración directa con HEIDI (y cualquier otra herramienta) con un impacto mínimo en su código.

Para hacer uso de este método de integración debe, desde el código de la aplicación a la que se pretenda integrar, ejecutarse la siguiente secuencia de operaciones:

1- Invocación de la facilidad de exportación de grafos (ya implementada en HEIDI o eventualmente a implementar si la herramienta utilizada no es HEIDI), generando un archivo que se almacena en un camino (path) dado.
2- Invocación al algoritmo adecuado, según lo desee el usuario puede puede optarse por el algoritmo de SIMPLIFICACIÓN, el algoritmo GRASP o el algoritmo genético. En esta invocación se indicará el camino (path) al archivo exportado en 1 y los parámetros propios del problema (uno de estos parámetros permite especificar el algoritmo a ejecutar). Esta invocación es extremadamente sencilla pues en esencia consta en una llamada al sistema operativo invocando la ejecución del motor de cálculo, pasándole a este los parámetros necesarios.
3- Importar desde HEIDI (en caso de utilizar HEIDI esta funcionalidad ya se encuentra implementada, en caso de utilizar otra puede ser necesario su implementación) el archivo de grafo resultado, el cual se encontrará ubicado en el mismo camino (path) que el archivo grafo de entrada especificado en el paso 2-.
4- Frente al caso en que se deseen utilizar (en HEIDI o en la herramienta correspondiente) los resultados relativos a la Diámetro-Confiabilidad obtenidos y al desempeño de nuestros algoritmos es posible importar información estadística. Esta información estadística se corresponde con la recabada por nuestros algoritmos durante el procesamiento en su búsqueda de la mejor solución para el problema y es generada en formato texto por nuestro “motor de cálculo”, siendo colocada en el mismo path que fuera especificado en el paso 2-.
5- Desplegar o manipular (en HEIDI o en la herramienta correspondiente) la información obtenida en los pasos 3- y 4-.

Aquí presentamos una representación gráfica de esta integración con HEIDI (o cualquier otra herramienta), los números indicados se corresponden con los pasos mencionados anteriormente:



Capítulo 6
Resultados Experimentales

6. Resultados Experimentales

6.1. Pruebas definidas

Las topologías de las diferentes redes de prueba fueron construidas respetando la compatibilidad con la herramienta HEIDI, permitiendo el uso de las funcionalidades provistas por dicha herramienta en caso de ser necesario.

Al definir las pruebas que llevaríamos a cabo definimos una estrategia que consistía en diferentes etapas.

Etapa 0: Etapa de Validación.

En esta etapa se complementó el testeo unitario realizado a lo largo de la fase de implementación, llevándose a cabo nuevas pruebas del tipo testeo unitario, modular y pruebas de stress para asegurar el uso óptimo de los recursos computacionales.

Etapa 1: Potencia del Algoritmo Simplificación.

El objetivo de esta etapa fue identificar juegos de pruebas que permitieran mostrar el desempeño del algoritmo Simplificación desarrollado. No se desarrollaron casos específicos para esta etapa sino que se hizo uso de los casos generales que serían usados en etapas posteriores, los mismos serán presentados en esta sección.

Para mostrar este desempeño se consideraron dos dimensiones:

1. El impacto en el tiempo total que ocasiona la ejecución del algoritmo Simplificación para cualquier grafo previo a la ejecución del algoritmo que se pretenda utilizar.
2. Medir de alguna forma la probabilidad de que al ejecutar el algoritmo Simplificación se obtenga la eliminación de aristas y/o nodos. Para ello se ejecuto sobre la base de todas las familias disponibles y se graficó un comparativo entre Casos Simplificados vs. Casos No Simplificados.

Etapa 2: Resultados de algoritmos vs. Solución óptima

Como no disponemos de una forma de obtener la solución exacta para un caso de prueba cualquiera, resultó imposible la comparación entre las soluciones de nuestros algoritmos y la solución óptima. Por tal razón esta etapa se hizo restringida a una clase de grafos en la que se conoce la solución óptima al problema.

Sea entonces la siguiente propiedad:

Propiedad Cancela-Petingi

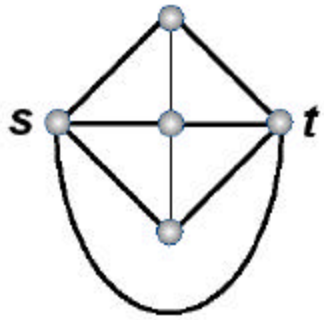
Dado un grafo G=(V, E, ψ) completo, para el que se cumple

i. $\forall$ e $\in$ E Costo(e)=b
ii. $\forall$ e $\in$ E Confiabilidad(e)=c
iii. $\forall$ e $\in$ E Largo(e)=l

Entonces, la red $G_{Rst}=(V_{Rst}, E_{Rst}, \psi_{Rst})$ que maximiza la Diámetro-Confiabilidad con D=2 y B=b(2(n-2)+1) cumple las siguientes propiedades:

a. $e^*=(s,t) \in V_{Rst}$
b. $V_{Rst}=V$
c. $\forall$ v $\in$ V_{Rst} con $v \neq s$ y $v \neq t$ se cumple que $e_1=(s, v) \in E_{Rst}$ y $e_2=(v, s) \in E_{Rst}$

Ejemplo gráfico:



Se muestra una red en la que las aristas resaltadas son las pertenecientes a caminos s-t que respetan la restricción de diámetro, para las demás aristas no existen caminos s-t que respeten dicha restricción y que las utilicen. Las aristas no resaltadas no contribuyen a la Diámetro-Confiabilidad.

Sobre la base de esta propiedad diseñamos una serie de pruebas que permitirían confirmar el correcto desempeño de los algoritmos.
Las pruebas se realizaron para un total de 6 grafos, según el siguiente detalle:

- grafos chicos (entre 1 y 40 nodos): 10 y 25 nodos.
- grafos medianos (entre 40 y 70 nodos): 55 y 68 nodos.
- grafos grandes (entre 70 y 100 nodos): 80 y 91 nodos.

Se implementó un algoritmo que confirmaría de forma automática si la topología de la solución obtenida luego de la ejecución se trataba de la esperada o no.

Etapa 3: Comparación entre algoritmos.
El objetivo de esta etapa consistió en la generación de un juego completo de pruebas que aseguraran una comparación relativa entre los algoritmos desarrollados. Se identificaron 108 familias de redes (grafos) las cuales serían utilizadas para la ejecución de ambos algoritmos.

El juego de pruebas finalmente utilizado fue generado de forma aleatoria según las características deseables para los grafos de las familias identificadas. Para evitar casos singulares se realizaron 5 casos de prueba por cada familia.

Durante el proceso de definición de las familias deseadas se identificaron cuales serían las dimensiones a tener en cuenta para las redes correspondientes. Cada familia resultó de una combinación posible entre los valores asociados a cada dimensión.

Dimensión 1:

Nro de Nodos. Los rangos posibles para el número de nodos fueron definidos sobre la base de las pruebas preliminares que realizamos, pruebas en las que consideramos el tiempo de ejecución y el tiempo total empleado. De este estudio, surge como rango base [10,100], del cual a su vez surgieron tres sub-rangos: [10, 40), [40,70) y [70, 100].

Para cada sub-intervalo, al generar cada grafo correspondiente (cantidad de nodos perteneciente al sub-intervalo) se sortearía un número entero en el intervalo asociado.

Dimensión 2:

Tipos de aristas. Los tipos posibles de aristas fueron:

- aristas con confiabilidad, largo y costo constantes e iguales a 0.5, 1, y 1 respectivamente
- aristas con confiabilidad, largo y costo variables y aleatorios definidos sobre los siguientes intervalos respectivamente (0, 1), [0, 10] y [1, 100].

Durante la ejecución de las pruebas se detectó la necesidad de disminuir la confiabilidad asignada a las aristas, ya que, los resultados que se obtenían convergían rápidamente a soluciones de Diámetro-Confiabilidad 1, dificultando la comparación de resultados. Se asignaron valores de confiabilidad en el intervalo (0, 0.2) para el caso de confiabilidades variables y 0.05 para el caso de confiabilidades constantes.

Dimensión 3:

Nro de aristas. Los valores posibles para el número de aristas fueron definidos sobre la base de la cantidad de nodos. Se identificaron dos posibles valores, siendo estos: $2*n$ y $n^2/4$.

Dimensión 4:

Diámetro. En lo que respecta a la elección de los sub-rangos asociados con el diámetro se vio la necesidad de que los mismos deberían estar íntimamente relacionados con la cantidad de nodos del grafo. Por tal motivo, consideramos que el mínimo diámetro a considerar en un grafo debía ser 2, ya que 1 tiene como solución trivial el grafo que tiene la arista que vincula s y t; y que el máximo diámetro debería darse como el largo del camino más largo posible, esto es n-1 (con n cantidad de nodos). Identificado el rango base de variación para el diámetro [2,n-1], se particionó en tres sub-rangos: [2, 2+(n-3)/3), [2+(n-3)/3, 2+2(n-3)/3) y [2+2(n-3)/3, n-1]. En cada caso, al generar un grafo cuyo diámetro estuviera en uno de estos tres sub-rangos se sortearía un número real en el intervalo asociado.

Dimensión 5:

Presupuesto. La identificación del rango base para el presupuesto debía estar relacionada con el Costo total del Grafo, el nro de aristas del grafo y de cierta forma el costo de cada arista. Por tal motivo se definiò el rango base como [Kmin, Kmax] con Kmin=Grafo.NroAristas * CostoMedioAristas/3 y Kmaxi=Grafo.Costo, del cual se obtuvo los subrangos: [Kmin, (Kmax - Kmin)/3 + Kmin][(Kmax - Kmin)/3 + Kmin, (Kmax - Kmin)*2/3 + Kmin] y [(Kmax - Kmin)*2/3 + Kmin, Kmax]

Una vez decididas estas dimensiones se obtienen 108 familias de prueba. Las familias quedan identificadas según el siguiente detalle:

- Nro. Nodos: 3 tipos de familias
- Tipo de aristas: 2 tipos de familias
- Nro. de aristas: 2 tipos de familias
- Diámetro: 3 tipos de familias
- Presupuesto: 3 tipos de familias

Por lo que se tienen 108 familias. Para cada familia se llevaron a cabo 5 pruebas (5 grafos aleatorios diferentes), resultando un total de 540 pruebas. Sobre cada uno de estas pruebas se llevo a cabo una ejecución del algoritmo GRASP y otra del algoritmo GA. El algoritmo creado para generar una red aleatoria toma el número de nodos y aristas deseadas, y determina las aristas que integran la red sorteando pares de vértices.

Debido a que se quería comparar algoritmos basados en metaheurísticas diferentes fue necesario equiparar posibilidades de ambos para asegurar la verosimilitud de las conclusiones que se obtuvieran. Definimos el concepto de unidades de procesamiento para cada algoritmo: para GRASP como la cantidad de iteraciones; para GA como el producto entre la cantidad de evoluciones y la cantidad de individuos en una población dada de GA.

Este planteamiento es razonable pero no es única la elección de los valores para cada uno de los factores que definen las unidades de procesamiento en GA. En ese sentido, decidimos balancear la cantidad de evoluciones y la cantidad de individuos optando por la raíz cuadrada de la cantidad de iteraciones de GRASP para cada uno de ellos.

6.2. Recolección de Resultados

6.2.1. Criterios

Se hicieron scripts para realizar de forma automática la recopilación de resultados, ya que de otra forma esto hubiera sido extremadamente costo dado el volumen de pruebas realizadas. Se puede referenciar los scripts realizados en la documentación que se adjunta a este documento.

Durante la ejecución de los algoritmos se generan información de salida que resumen datos estadísticos. Esta salida contiene: unidad de procesamiento en la cual se genera el reporte, tiempo transcurrido hasta el reporte, unidad de procesamiento dónde se encontró la mejor solución hasta el reporte, tiempo insumido para encontrar dicha solución, intervalo de confianza de la solución, número de replicaciones asociado al cálculo de la Diámetro-Confiabilidad con MCC y el grafo solución; los cuales fueron separados por comas "," para facilitar la generación de planillas. A los efectos de la unicidad de los reportes a la hora de consolidar, definimos un sufijo en el nombre de los archivos: GA para Algoritmos Genéticos y GR para GRASP.

El formato para el nombre del archivo de grafo de entrada fue GFFFPP.exp, donde FFF es el número de familia correspondiente y PP el número de prueba. Los grafos de salida tienen el formato GFFFPPAAN.exp, donde AA representa el algoritmo utilizado (GR=GRASP, GA=Algoritmo Genético) y N representa el número de reporte.

Los reportes fueron realizados en tres instancias, de forma de valorar y comparar entre los algoritmos la calidad de identificación de la mejor solución con poco, mediano y total de unidades de procesamiento dado. Se definió un número de unidades de procesamiento máxima para cada caso y se realizó el reporte a la mitad, tres cuarta parte y al total de unidades de procesamiento. Cada reporte fue identificado con un ordinal el cual se agregó como sufijo al nombre de los archivos.

De esta forma, en una ejecución de GRASP sobre el grafo G10805.exp, tendríamos 6 archivos:

- G10805GR1.exp, G10805GR1.txt correspondiente al primer reporte realizado a la mitad del total del procesamiento.
- G10805GR2.exp, G10805GR2.txt correspondiente al segundo reporte realizado a la tres cuarta parte del total del procesamiento.
- G10805GR3.exp, G10805GR3.txt correspondiente al tercer y último reporte realizado al completarse la totalidad del procesamiento.

No obstante esta regla en la definición de los reportes para los algoritmos, definimos a nivel del código fuente sentencias de escape, las cuales tras identificar alguna condición especial abortarían la ejecución de cualquiera de los algoritmos. A saber:

- la identificación de una solución de máxima Diámetro-Confiabilidad, según las restricciones impuestas por el número máximo de replicaciones dado para el MCC.
- La identificación de que el costo de incluir todas las aristas posibles del Grafo simplificado fuera menor o igual que el presupuesto dado.

En lo que hace referencia a descartar resultados particulares que pudieran desvirtuar la correcta interpretación de la generalidad de los casos, indicamos que no fue necesario tomar acción alguna.

Las comparaciones realizadas tienen en cuenta el número de iteraciones del algoritmo GRASP contra el número de evoluciones y la cantidad de individuos en la población del algoritmo GA.
Los puntos tenidos en cuenta en la comparación relativa fueron los siguientes:

- La Diámetro-Confiabilidad máxima obtenida al final del procesamiento
- Unidades de procesamiento empleadas para alcanzar ese máximo
- Evolución en la búsqueda de la solución remitiéndonos al log de cada proceso
- Evolución en la búsqueda de la solución en base a los tres reportes.
- Variaciones en el tiempo total de procesamiento al incrementarse el tamaño de las pruebas

Entendemos que el nivel de detalle con el cual se haga necesario el estudio de las pruebas dependerá de la dificultad o no de identificar patrones de comportamiento en el desempeño de los algoritmos.

6.2.2. Resultados Obtenidos

En esta sección presentamos los resultados obtenidos durante la fase de pruebas discriminado por etapa, según lo expuesto en la Sección 6.1 *Criterios para las Pruebas*.

Etapa 0: Etapa de Validación.

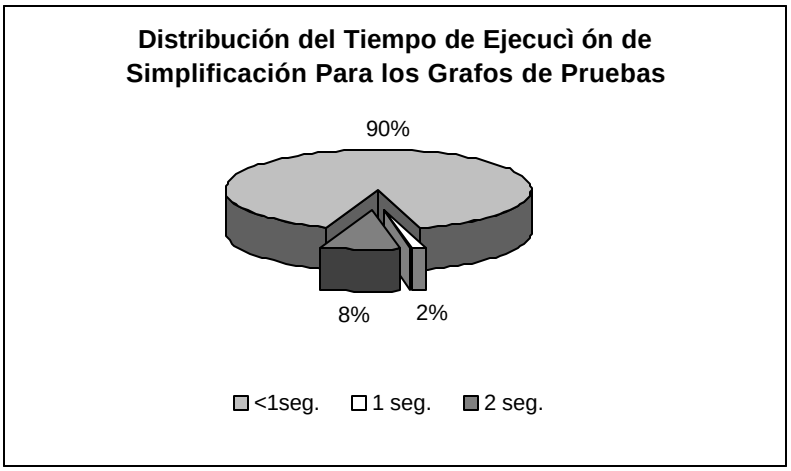
Esta etapa no generó resultados que tengan sentido en esta sección.

Etapa 1: Potencia del Algoritmo Simplificación.

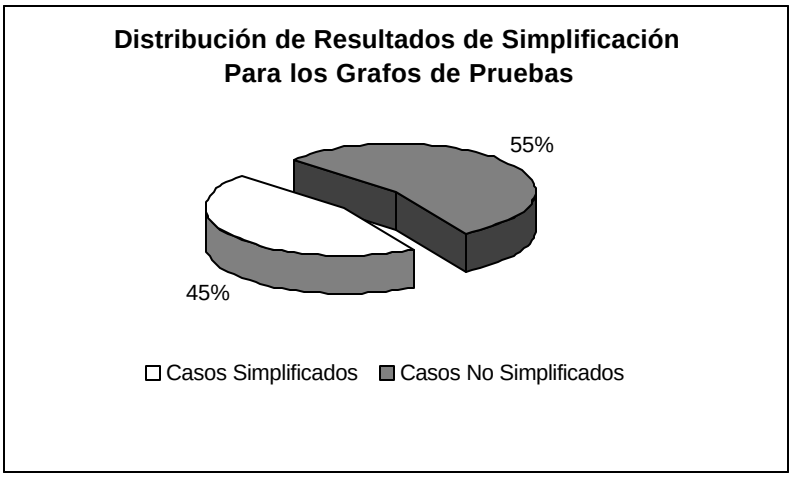
El objetivo de esta etapa fue identificar juegos de pruebas que permitieran mostrar el desempeño del algoritmo Simplificación desarrollado considerando la restricción de diámetro. No se consideraron las pruebas que diferían solo en el presupuesto, por lo que en suma se llevaron a cabo 180 casos de prueba ([#Nodos]*[#Tpo aristas]*[#Aristas]*[Diám]=3*2*2*3). Las redes de prueba se generaron aleatoriamente según la familia correspondiente, siendo cada red generada en forma enteramente independiente de las demás. Como se indicó el juego de pruebas fue el mismo utilizado para las restantes etapas. La planilla con la información de ejecución aparece en la Sección 9.5.4 *Archivos Resultados.*

De la ejecución de las pruebas resultan las siguientes consideraciones:

1. El tiempo total insumido para la ejecución fue del orden de 33 segundos, se puede ver que el tiempo promedio es del orden de dos décimas de segundo al haber 180 casos de prueba. De esta forma, la ejecución de Simplificación no afecta en absoluto el tiempo total de la ejecución del algoritmo, por lo que se recomienda incluirlo como parte de cualquier algoritmo para el problema de la Diámetro-Confiabilidad.



2. El desempeño al eliminar arista y/o nodos fue medio en la medida que no siempre se realizó una eliminación.



Además cuando se produjo una simplificación la misma fue moderada, del orden del 7% en promedio para la simplificación de nodos y de 3% para aristas. La planilla con la información de ejecución aparece en la Sección 9.5.4 *Archivos Resultados.*

Etapa 2: Algoritmos vs. Real

En todos los casos se obtuvo la topología esperada, por lo que se concluye que el desempeño de los algoritmos es el correcto frente a esta clase de problema. La planilla con la información de ejecución aparece en la Sección 9.5.4 *Archivos Resultados.*

En estas pruebas se utilizó el algoritmo de Simplificación y el tamaño de los grafos utilizados fue de 10, 25, 55, 68, 80 y 91 nodos, siendo grafos completos donde las aristas tienen confiabilidad constantes y costo y largo unitarios. El número de iteraciones utilizados para GRASP fue respectivamente 25, 25, 64, 64, 100, 100; y en el caso de GA se utilizaron como tamaño de población los valores 5, 5, 8, 8,10 y 10 y como número de generaciones 5, 5, 8, 8, 10 y 10.

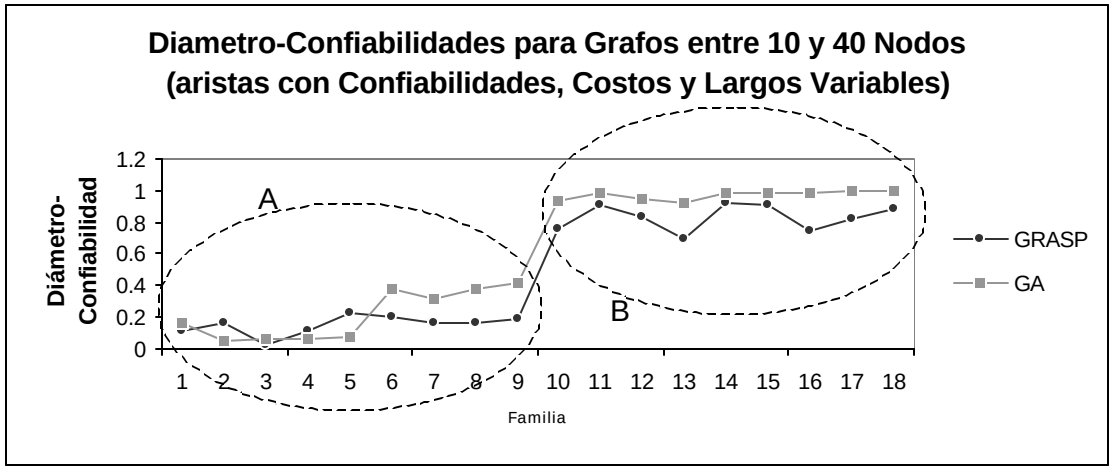
Etapa 3: Comparación entre Algoritmos
El objetivo de esta etapa consistió en la generación de un juego completo de pruebas que aseguraran una comparación relativa entre los algoritmos desarrollados. De esta etapa destacamos loa siguientes resultados.

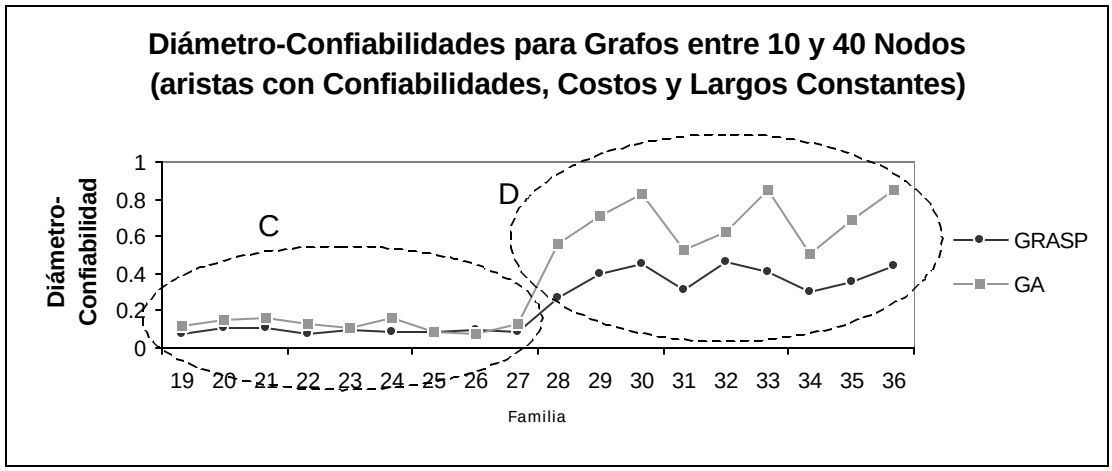
1) Confiabilidades Obtenidas por tipos de grafo

Debe tenerse en cuenta que como valor representativo de las confiabilidades obtenidas por cada algoritmo se utilizó el extremo superior del intervalo de confianza en el que se encuentra el valor de la confiabilidad.

A continuación presentamos los máximos obtenidos por cada algoritmo de acuerdo a las pruebas. Los máximos están indicados en cada una de las planillas para cada familia. Los mismos representan el extremo superior del intervalo de confiabilidad máximo encontrado por el algoritmo de superior desempeño.

Grafos entre 10 y 40 nodos





En base al comportamiento de los algoritmos para las familias correspondientes a redes con un número de nodos en el rango [10, 40] hemos identificado, dentro de una elipse, cada sección que consideramos relevante. Las familias 1 a 36 conforman este grupo.

A. Los casos de pruebas agrupados por estas familias presentan las siguientes características:
   - Costos variables en las aristas, elegidos al azar.
   - Largos variables en las aristas, elegidos a azar.
   - Confiabilidades variables en las aristas, elegidas al azar.
   - Relación entre nodos y aristas dada por: #Aristas=2 * #Nodos
   - Las pruebas realizadas cubren todo el rango base para el diámetro.
   - Las pruebas realizadas cubren todo el rango base para el presupuesto.

En este caso puede verse que las soluciones encontradas por cada algoritmo siguen el mismo patrón de calidad, donde por momentos las soluciones algoritmo GRASP tienen una calidad mayor a las soluciones algoritmo GA.

B. Los casos de pruebas agrupados por estas familias presentan las siguientes características:
   - Costos variables en las aristas, elegidos al azar.
   - Largos variables en las aristas, elegidos a azar.
   - Confiabilidades variables en las aristas, elegidas al azar.
   - Relación entre nodos y aristas dada por: #Aristas=#Nodos^2/4
   - Las pruebas realizadas cubren todo el rango base para el diámetro.
   - Las pruebas realizadas cubren todo el rango base para el presupuesto.

En este caso puede verse que algoritmo GA tiene la supremacía en la calidad de las soluciones y que las calidades de las soluciones de ambos algoritmos presentan la misma tendencia.

C. Los casos de pruebas agrupados por estas familias presentan las siguientes características:
   - Costos constantes en las aristas.
   - Largos constantes en las aristas.
   - Confiabilidades constantes en las aristas.
   - Relación entre nodos y aristas dada por: #Aristas=2 * #Nodos
   - Las pruebas realizadas cubren todo el rango base para el diámetro.
   - Las pruebas realizadas cubren todo el rango base para el presupuesto.

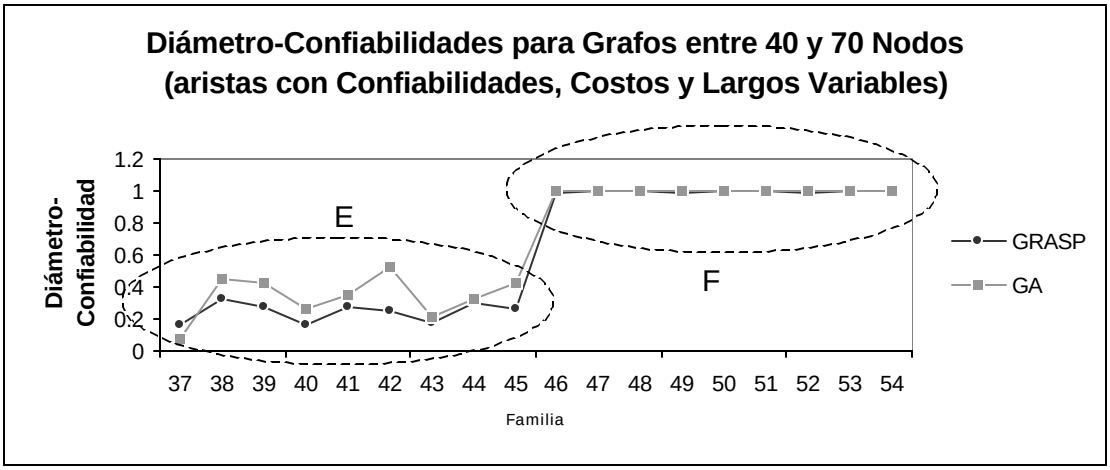
En este caso puede verse que el algoritmo GA tiene la supremacía en la calidad de las soluciones en prácticamente todas estas familias, nuevamente las calidades de las soluciones de ambos algoritmos presentan la misma tendencia.

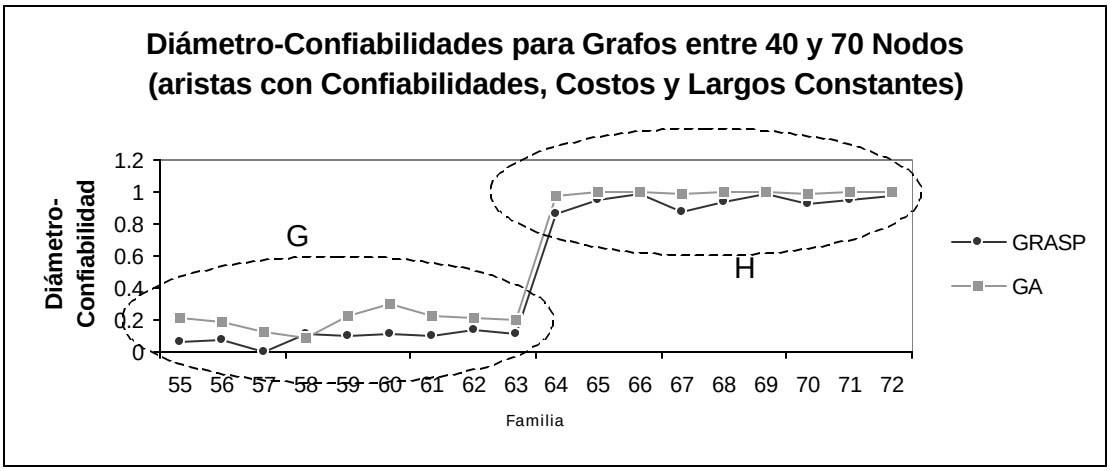
D. Los casos de pruebas agrupados por estas familias presentan las siguientes características:
   - Costos constantes en las aristas.
   - Largos constantes en las aristas.
   - Confiabilidades constantes en las aristas.
   - Relación entre nodos y aristas dada por: #Aristas=#Nodos^2/4
   - Las pruebas realizadas cubren todo el rango base para el diámetro.
   - Las pruebas realizadas cubren todo el rango base para el presupuesto.

   En este caso puede verse que el algoritmo GA tiene la supremacía en la calidad en todas estas familias involucradas; las calidades de las soluciones de ambos algoritmos siguen un patrón similar, destacándose la mayor separación en las calidades de las soluciones de este grupo de familias.

Conclusiones para este grupo de familias:
En este grupo de familias se puede detectar que en redes pequeñas el algoritmo GRASP tiene un mejor desempeño que el algoritmo GA.
Las calidades de las soluciones de ambos algoritmos siguen un patrón similar. En los casos en que el número de aristas es mayor (B y D) tiene lugar la mayor diferencia entre el algoritmo GA y el algoritmo GRASP.

Grafos entre 40 y 70 nodos





En base al comportamiento de los algoritmos para las familias correspondientes a redes con un número de nodos en el rango [40, 70] hemos identificado, dentro de una elipse, cada sección que consideramos relevante. Las familias 37 a 72 conforman este grupo.

E. Los casos de pruebas agrupados por estas familias presentan las siguientes características:
   - Costos variables en las aristas, elegidos al azar.
   - Largos variables en las aristas, elegidos a azar.
   - Confiabilidades variables en las aristas, elegidas al azar.
   - Relación entre nodos y aristas dada por: #Aristas=2 * #Nodos
   - Las pruebas realizadas cubren todo el rango base para el diámetro.
   - Las pruebas realizadas cubren todo el rango base para el presupuesto.

En este caso puede verse que las soluciones encontradas por cada algoritmo siguen el mismo patrón de calidad, donde en un único caso la solución dada por el algoritmo GRASP tiene una calidad mayor a la correspondiente del algoritmo GA.
En general, el algoritmo GA tiene un desempeño superior al algoritmo GRASP.

F. Los casos de pruebas agrupados por estas familias presentan las siguientes características:
   - Costos variables en las aristas, elegidos al azar.
   - Largos variables en las aristas, elegidos a azar.
   - Confiabilidades variables en las aristas, elegidas al azar.
   - Relación entre nodos y aristas dada por: $\#Aristas=\#Nodos^2/4$
   - Las pruebas realizadas cubren todo el rango base para el diámetro.
   - Las pruebas realizadas cubren todo el rango base para el presupuesto.

En este caso puede verse que el algoritmo GA encontró soluciones óptimas mientras que el algoritmo GRASP en la mayoría de los casos encontró soluciones de menor calidad, aunque muy cercanas a la óptima. Es importante destacarlo en virtud de que, en estos casos, las confiabilidades asignadas a las aristas son bajas, elegidas en el intervalo (0, 0.2).

G. Los casos de pruebas agrupados por estas familias presentan las siguientes características:
   - Costos constantes en las aristas.
   - Largos constantes en las aristas.
   - Confiabilidades constantes en las aristas.
   - Relación entre nodos y aristas dada por: $\#Aristas=2 * \#Nodos$
   - Las pruebas realizadas cubren todo el rango base para el diámetro.
   - Las pruebas realizadas cubren todo el rango base para el presupuesto.

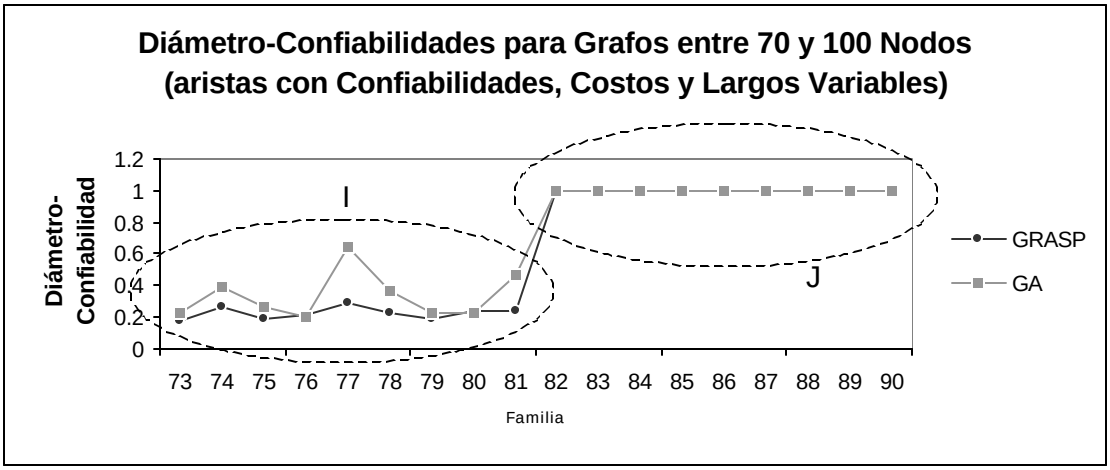
En este caso puede verse que el algoritmo GA tiene la supremacía en la calidad de la mayoría de las soluciones, siendo superado por el algoritmo GRASP en únicamente un caso.

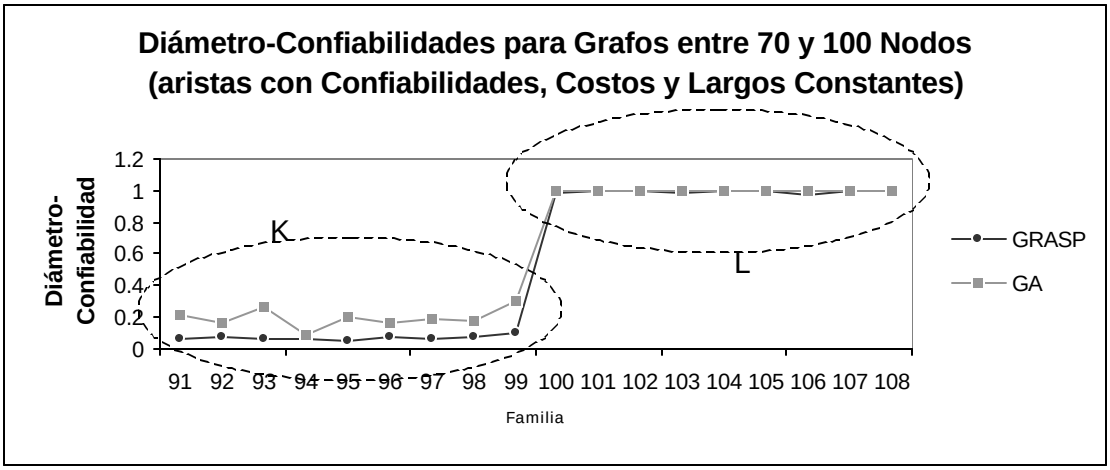
H. Los casos de pruebas agrupados por estas familias presentan las siguientes características:
   - Costos constantes en las aristas.
   - Largos constantes en las aristas.
   - Confiabilidades constantes en las aristas.
   - Relación entre nodos y aristas dada por: $\#Aristas=\#Nodos^2/4$
   - Las pruebas realizadas cubren todo el rango base para el diámetro.
   - Las pruebas realizadas cubren todo el rango base para el presupuesto.

En este caso puede verse que nuevamente el algoritmo GA tiene la supremacía en la calidad de soluciones, teniendo confiabilidades prácticamente constantes y a menos de $2*10^{-2}$ de la solución máxima posible. El algoritmo GRASP tiene un comportamiento más irregular.

Conclusiones para este grupo de familias:
El algoritmo GA encuentra soluciones a menos de $2*10^{-2}$ de la solución máxima posible para el caso de pruebas con un número de aristas igual a $\#Nodos^2/4$, sin importar la confiabilidad asignada a las aristas (F y H). Por el contrario GRASP encuentra soluciones cercanas al óptimo para estos mismos casos, pero se ve fuertemente afectado por la diferencia en la asignación de confiabilidad de las aristas (en F confiabilidades en (0, 0.2) y en H confiabilidades con valor 0.05).
El algoritmo GA tiene un desempeño superior al algoritmo GRASP en la mayoría de los casos.

Grafos entre 70 y 100 nodos





En base al comportamiento de los algoritmos para las familias correspondientes a redes con un número de nodos en el rango [70, 100] hemos identificado, dentro de una elipse, cada sección que consideramos relevante. Las familias 73 al final (108) conforman este grupo.

I. Los casos de pruebas agrupados por estas familias presentan las siguientes características:
   - Costos variables en las aristas, elegidos al azar.
   - Largos variables en las aristas, elegidos a azar.
   - Confiabilidades variables en las aristas, elegidas al azar.
   - Relación entre nodos y aristas dada por: #Aristas=2 * #Nodos
   - Las pruebas realizadas cubren todo el rango base para el diámetro.
   - Las pruebas realizadas cubren todo el rango base para el presupuesto.

   En este caso ambos algoritmos encuentran soluciones de calidades comparables, teniéndose algunas excepciones, en los que el algoritmo GA tiene un desempeño notablemente superior al algoritmo GRASP.

J. Los casos de pruebas agrupados por estas familias presentan las siguientes características:
   - Costos variables en las aristas, elegidos al azar.
   - Largos variables en las aristas, elegidos a azar.
   - Confiabilidades variables en las aristas, elegidas al azar.
   - Relación entre nodos y aristas dada por: #Aristas=#Nodos^2/4
   - Las pruebas realizadas cubren todo el rango base para el diámetro.
   - Las pruebas realizadas cubren todo el rango base para el presupuesto.

   En este caso puede verse que tanto el algoritmo GA como el algoritmo GRASP encuentran soluciones de confiabilidad 1.

K. Los casos de pruebas agrupados por estas familias presentan las siguientes características:
   - Costos constantes en las aristas.
   - Largos constantes en las aristas.
   - Confiabilidades constantes en las aristas.
   - Relación entre nodos y aristas dada por: #Aristas=2 * #Nodos
   - Las pruebas realizadas cubren todo el rango base para el diámetro.
   - Las pruebas realizadas cubren todo el rango base para el presupuesto.

   En este caso GA tiene la supremacía en la calidad de las soluciones, teniendo un desempeño comparable al de GRASP en un único caso.

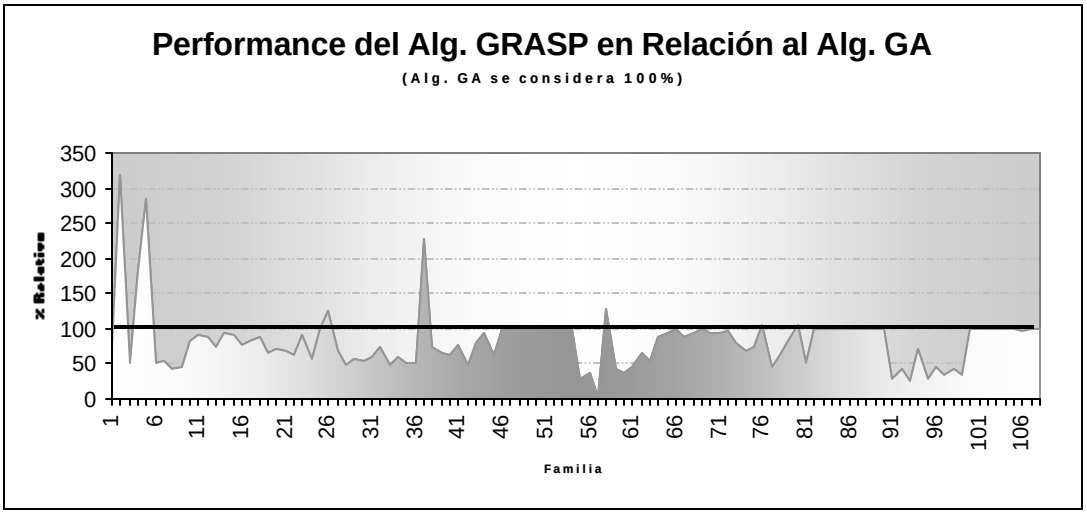
L. Los casos de pruebas agrupados por estas familias presentan las siguientes características:
   - Costos constantes en las aristas.
   - Largos constantes en las aristas.
   - Confiabilidades constantes en las aristas.
   - Relación entre nodos y aristas dada por: #Aristas=#Nodos^2/4
   - Las pruebas realizadas cubren todo el rango base para el diámetro.
   - Las pruebas realizadas cubren todo el rango base para el presupuesto.

Nuevamente, el algoritmo GA presenta un desempeño superior al algoritmo GRASP en todos los casos. Recordamos que en este caso se trata de aristas asignadas con confiabilidad baja (constante 0.05).

Conclusiones para este grupo de familias:
El algoritmo GA encuentra soluciones cercanas al óptimo para el caso de pruebas con un número de aristas igual a #Nodos^2/4, sin importar la confiabilidad asignada a las aristas (J y L). Por el contrario, el algoritmo GRASP encuentra soluciones cercanas al óptimo para estos mismos casos, pero se ve afectado aunque mínimamente, por la diferencia en la asignación de confiabilidad de las aristas (en J confiabilidades en (0, 0.2) y en L confiabilidades con valor 0.05).
El algoritmo GA tiene un desempeño superior en la mayoría de los casos.

Conclusiones Globales:
El algoritmo GA presenta un mejor desempeño en la generalidad de los casos.
Dentro del conjunto de pruebas generados al azar fue visible un comportamiento en el que ambos algoritmos alcanzaron soluciones óptimas.
Las máximas confiabilidades fueron alcanzadas en los casos en que los grafos disponen del mayor número de aristas considerado para el conjunto de pruebas.
Se notó la calidad de los algoritmos al tener ambos la misma tendencia en general aunque el algoritmo GRASP con alguna característica irregular. En el caso I se da la mayor diferencia entre las calidades de las soluciones respectivas de cada algoritmo.

2) Comparativo de Calidad de Soluciones

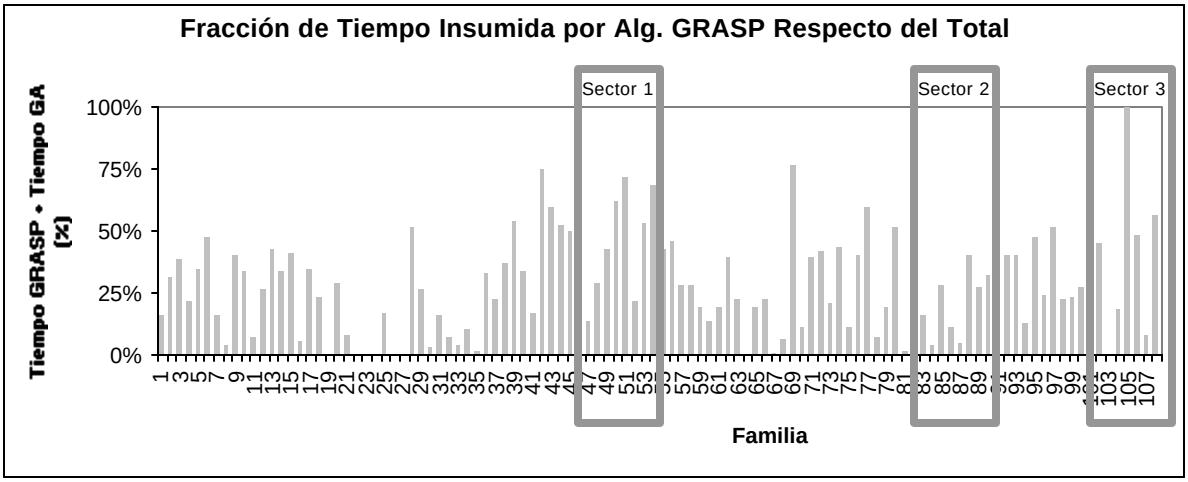


La grafica anterior compara la calidad de la solución encontrada por el algoritmo GRASP frente a la correspondiente encontrada por el algoritmo GA. En la misma se grafica, de forma porcentual, el valor obtenido por el algoritmo GRASP (como extremo derecho del intervalo de confianza para la Diámetro-Confiabilidad) con respecto al valor correspondiente obtenido por el algoritmo GA, el cual se considera como el 100%.

De la misma se extraen las siguientes observaciones:

- Se visualiza que los valores graficados están mayoramente por debajo del 100%, indicando la inferioridad de las soluciones encontradas por el algoritmo GRASP frente a las obtenidas por el algoritmo GA.
- Una vez más se visualiza los grupos de familias en las cuales el algoritmo GA y el algoritmo GRASP obtuvieron soluciones con valores similares de Diámetro-Confiabilidad. Simultáneamente remarca situaciones en dónde se muestran grandes diferencias en la calidad de la solución obtenida por cada uno de los algoritmos, identificándose casos en los que uno prima sobre el otro y viceversa.

3) Fracción de tiempo insumida por Algoritmo



La grafica incluida anteriormente se hizo sobre la base de los tiempos insumidos por los algoritmos y en ella solo hemos querido hacer énfasis en algunas zonas, remarcando de forma particular las mismas.

Esta gráfica representa comparativamente las fracciones de tiempo insumidas por cada algoritmo respecto del total. El área total de la gráfica representa el tiempo de procesamiento total insumido en la fase de pruebas. El área en gris representa la fracción de tiempo total insumido por el algoritmo GRASP en la fase de pruebas.

Cada sector remarcado se corresponde con las pruebas en las que ambos algoritmos alcanzaron soluciones óptimas o muy cercanas a la solución óptima. Visualizando individualmente cada sector es posible obtener una comparación de los tiempos empleados por cada algoritmo, teniendo en cuenta que el área en gris representa la fracción de tiempo insumido por el algoritmo GRASP en las familias correspondientes.

Los tiempos resultan comparables en el primer y tercer sector, mientras que en el segundo resulta evidente que el algoritmo GA ha consumido un tiempo excesivo, en comparación con el algoritmo GRASP, para alcanzar soluciones de calidad equiparable.

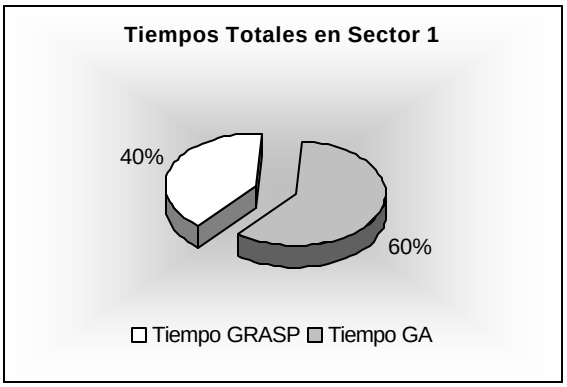
A continuación presentamos un comparativo de tiempos totales, tanto para la fase total de pruebas como para cada uno de los sectores remarcados.

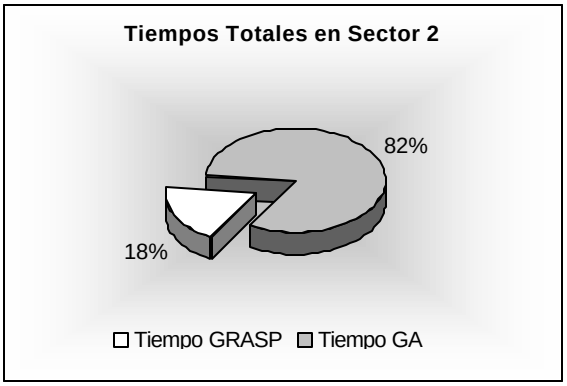
Sector 1
Familias : 46 a la 54
Tiempo Alg. GRASP : 12201 seg.
Tiempo Alg. GA : 18217 seg.

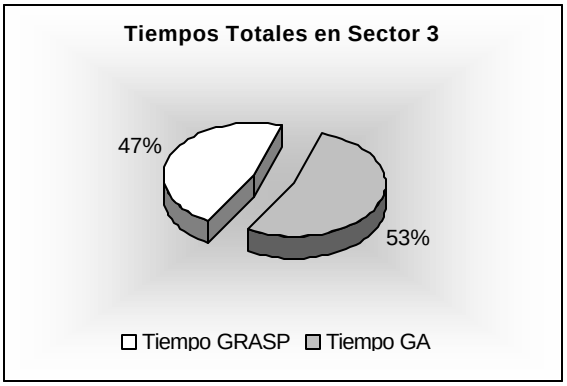
Sector 2
Familias : 82 a la 90
Tiempo Alg. GRASP : 2245 seg.
Tiempo Alg. GA : 10222 seg.

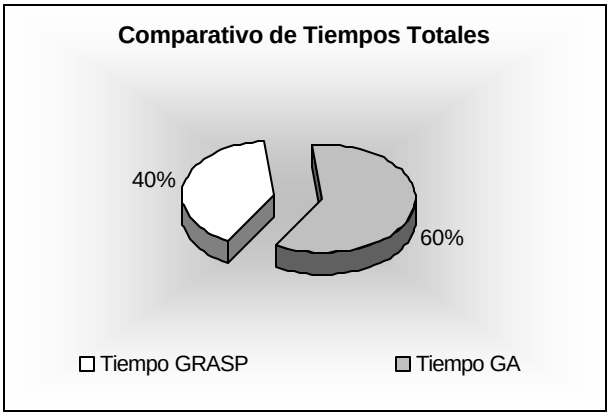
Sector 3
Familias : 100 a la 108
Tiempo Alg. GRASP : 291833 seg.
Tiempo Alg. GA : 334783 seg.

Fase de Pruebas:
Familias : 1 a la 108
Tiempo Alg. GRASP : 358641 seg.
Tiempo Alg. GA : 535025 seg.

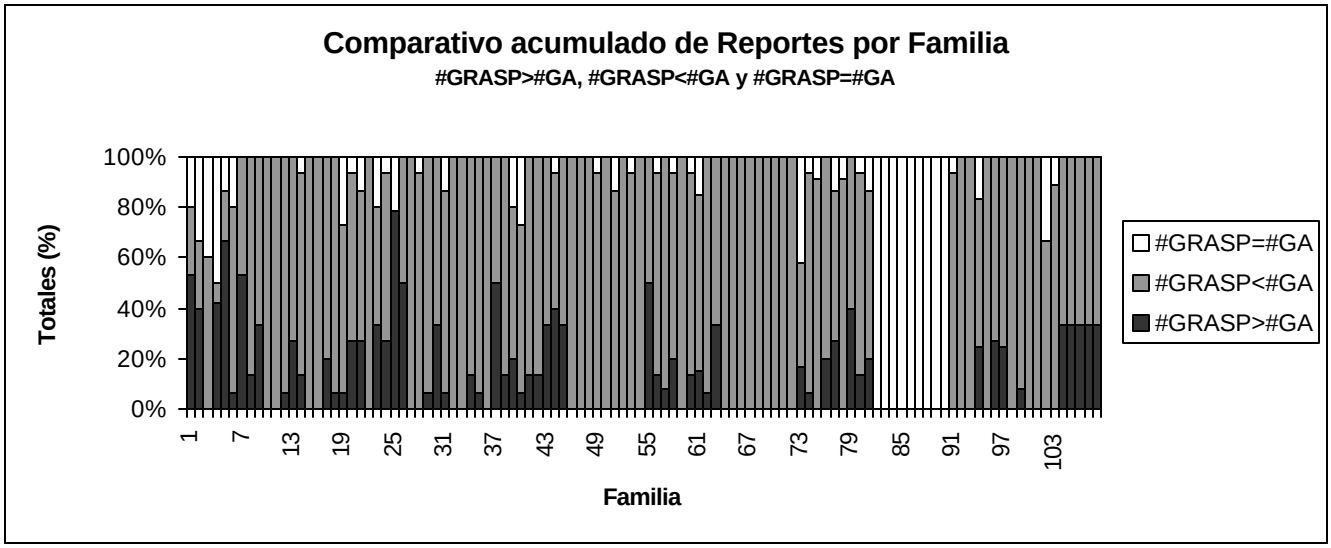






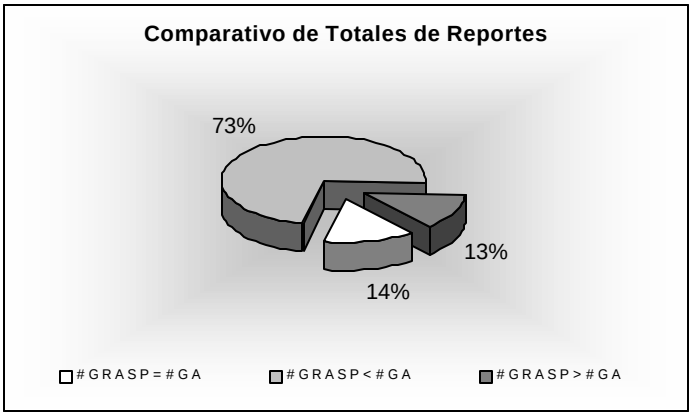


4) Comparativo acumulado de Reportes por Familia



Al realizar el estudio de los intervalos de confianza reportados por los algoritmos pudimos ver que en la generalidad de los casos los mismos son disjuntos, solo detectándose una intersección en aquellos casos en los que se alcanzó la solución de confiabilidad 1. Por lo tanto resultó válido utilizar el extremo superior del intervalo de confianza como medida para cada una de las soluciones encontradas, y considerar únicamente como casos "#GRASP=#GA" aquellos en que el algoritmo GRASP y el algoritmo GA alcanzaron exactamente el mismo valor de extremo superior de intervalo, los cuales solo ocurrieron al alcanzar soluciones de confiabilidad 1.

En este cuadro se resume el desempeño de cada algoritmo para cada una de las familias utilizadas en los casos de prueba. Lo que se grafica para cada algoritmo es, en porcentajes, cuanto representa el número de reportes que tuvo como resultado un valor de Diámetro-Confiabilidad superior, inferior e igual al obtenido por el otro algoritmo respecto del total de reportes realizados para cada familia. Se recuerda que fueron realizados 15 reportes por cada familia, número que resulta de 3 ejecuciones para cada uno de los 5 grafos que conforman la familia.



6.3. Resumen

Sobre la base de lo expuesto en la sección anterior, puntualizamos entonces que las conclusiones arribadas de acuerdo a los resultados obtenidos son:

1. El tiempo total insumido para la ejecución del algoritmo Simplificación es despreciable. La ejecución de Simplificación no afecta en absoluto el tiempo total de la ejecución del algoritmo, por lo que, es recomendable utilizarlo como parte de cualquier algoritmo para el problema de la Diámetro-Confiabilidad.
2. Los algoritmos implementados verifican un comportamiento correcto frente a la Propiedad “Propiedad Cancela-Petingi”.
3. El algoritmo GA presenta un mejor desempeño en la generalidad de los casos con respecto al algoritmo GRASP:
   - Obtiene mejores soluciones por unidad de procesamiento.
   - Describe una mejora acentuada a medida que transcurren unidades de procesamiento.
4. El algoritmo GRASP insume en la generalidad de los casos menos recursos computacionales que el algoritmo GA: tiempo de CPU y espacio de memoria. Por lo se concluye que sería oportuno realizar una prueba más justa de los algoritmos según el uso de los recursos computacionales en futuros trabajos.

Capítulo 7
Conclusiones Finales y Trabajos Futuros

7. Conclusiones Finales y Trabajos Futuros

7.1. Conclusiones Finales

Concluido el proyecto, corresponde resaltar los siguientes aspectos.

En lo que refiere a los Objetivos:

Se cumplieron todos los objetivos indicados en el Capítulo 2 *Objetivos*.

Además de los objetivos puntuales relativos al problema que se debía resolver, siempre tuvimos presentes otros adicionales, los cuales fueron cumplidos y los que por su repercusión en el resultado final merecen ser destacados:

- Trabajar fuertemente y con esmerado empeño en puntos que habitualmente no son visibles (códigos fuentes, validación de la solución, uso efectivo de memoria y de tiempo de procesador) de igual forma en la que se trabaja con aquellos puntos que si serían visibles.
- Definir y utilizar un método bien especificado y lo más amplio posible para la selección de las metaheurísitcas más adecuadas que permitiera, no solo cumplir con el objetivo impuesto de seleccionar las dos más adecuadas, sino que constituyera un método de referencia para otros trabajos afines.
- Llevar a cabo un trabajo que permitiera un fácil entendimiento, que simplificara el uso adecuado de los resultados de éste y favoreciera el desarrollo eficaz por parte de grupos de trabajo afines. Esto está particularmente reforzado por una documentación de pseudocódigos clara, correcta y entendible; códigos fuente con iguales características; secciones especialmente dedicadas al apoyo de otros grupos de trabajo; scripts que automatizan la generación de grafos aleatorios, el consolidado de datos y la generación de métricas asociadas; juegos de datos extensos que pueden referirse sin la necesidad de reejecuciones de los algoritmos.
- Realizar un trabajo, de excelente calidad, que se constituyera sin lugar a dudas como un aporte relevante a los campos de investigación relacionados al problema que debíamos resolver, proporcionando un material serio y confiable.

En lo que refiere a Puntualizaciones Generales:

El lenguaje de programación utilizado resultó especialmente adecuado para el desarrollo, la validación y el mantenimiento del código desarrollado.

La herramienta HEIDI presenta un diseño que permite una ampliación de sus funcionalidades a través de mecanismos simples y suficientemente generales para la integración con otro software.

La imposición en la práctica del uso de un método precario para el cálculo de la Diámetro-Confiabilidad de una red constituyó, entre otras cosas, una dificultad importante a la hora de llevar a cabo la implementación.

Los algoritmos desarrollados presentan un muy buen desempeño en la práctica. Comparativamente puede verse que el algoritmo GA obtiene, en la mayoría de los casos, las soluciones de mayor Diámetro-Confiabilidad y que simultáneamente requiere un tiempo de procesamiento mayor al empleado por el algoritmo GRASP.

7.2. Algunas Consecuencias de este Taller

Dado que este taller constituye la primera experiencia práctica aplicada al Diseño de Redes Diámetro-Confiables, aporta al campo de la confiabilidad en redes los primeros resultados empíricos para este problema.

Por otra parte, este trabajo destaca las dificultades que surgen en el esfuerzo de lograr algoritmos para resolverlo, dificultades propias del problema; así como las estrategias para la resolución de estas dificultades y del problema en general.

Los algoritmos implementados, además de materializar una solución al problema, son un valioso aporte en la medida que estarán disponibles para ser reutilizados en otros proyectos de investigación.

Por último, destacar que la investigación teórica puede verse guiada en algún sentido a través de las conjeturas, resultados y/o conclusiones que se brindan aquí.

7.3. Trabajos Futuros

Las tareas recomendadas para continuar este trabajo, las cuales apuntan a una investigación más profunda y a una ampliación de los aportes proporcionados por este taller son:

- Búsqueda exhaustiva de la combinación de valores posibles para los parámetros, con el objetivo de encontrar la combinación óptima. Para esto se hace necesario también encontrar la relación entre los parámetros y las características propias del grafo que se quiere resolver.
- Estudiar la resolución al problema del cálculo de la Diámetro-Confiabilidad en una red de forma óptima.
- Procesar y aplicar la lista de sugerencias funcionales propuestas, sugerencias que por razones de tiempo no fueron posibles implementar, pero que, por su relevancia, justifican su estudio e implementación.

Capítulo 8
Referencias

8. Referencias

[ADS1997] Fulya Altiparmak, Berna Dengiz, Alice E. Smith, *Local Search Genetic Algorithm for Optimal Design of Reliable Networks*, IEEE Transactions on Evolutionary Computation, August 1997.

[CDM+1991] A. Colorni, M.Dorigo, F. Maffioli, V. Maniezzo, G. Righini, M. Trubian, *Heuristics From Nature For Hard Combinatorial Optimization Problems*, Interantional Transactions in Operational Research, pp. 1-21.

[CEI1997] Departamento de Investigación Operativa, *Publicación "Grafos"*, CEI – Centro Estudiantes de Ingeniería, Facultad de Ingeniería – Universidad de la República, Montevideo, Uruguay (1997).

[CP2001a] Héctor Cancela y Luis Petingi, *Diameter constrained Network realiability: exact evaluation by factorization and bounds* , Proceedings of the International Conference on Industrial Logistics, Okinawa, Japan, (2001), pp.359-366.

[D1992] M. Dorigo, *Ph.D. thesis "Optimization, Learning and Natural Algorithms"*, Dipartimento di Elettronica, Politecnico di Milano, Italia 1992.

[DC1991] M. Dorigo, V. Maniezzo, y A. Colorni, *Positive feedback as a search strategy*, Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Italia 1991.

[DDG] M Dorigo, G. Di Caro, Luca M. Gambardella, *Ant Algorithms for Discrete Optimization*, Artificial Life, 1999, Vol.5, No.3, pp.137-172.

[DG1997a] M. Dorigo y L. M. Gambardella, *Ant colonies for the traveling salesman problem*, BioSystems, 1997, pp. 43:73 –81.

[DG1997b] M. Dorigo y L.M. Gambardella, *Ant colony system: A cooperative learning approach to the traveling salesman problem*, IEEE Transactions on Evolutionary Computation, 1997, pp. 1(1):53 –66.

[DS] Darren L.Deeter, Alice E. Smith, *Heuristic Optimization of Network Design Considering All-Terminal Reliability*, 97RM-060

[F1993] B. L. Fox, *Integrating and accelerating Tabu search, simulated annealing and Genetic Algorithms*, Annals of Operations Research 41, 1993, pp. 47-67.

[FK1992] U. Faigle y W. Kern, *Some Convergence Results for Probabilistic Tabu Search*, ORSA Journal on Computing 4, 1992, pp. 32-37.

[FR1989] T.A. Feo y M.G.C. Resende, *A probabilistic heuristic for a computationally difficult set covering problem*, Operations Research Letters, 1989, 8:67–71.

[FR1995] T.A. Feo y M.G.C, *Resende. Greedy randomized adaptive search procedures*, Journal of Global Optimization, 1995, 6:109–133.

[FS1992] James A. Freeman y David M. Skapura, *Neural Networks: Algorithms, Applications, and Programming Techniques*, Addison Wesley, July 1992, pp.89-126.

[G1986] F. Glover, *Future Paths for Integer Programming and Links to Artificial Intelligence Computers and Operations Research*, 1986, pp. 533-549.

[G1989a] D.E Goldberg, *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley, 1989.

[G1989b] F. Glover, *Tabu search – Part I*, ORSA Journal on Computing, 1:190–206, 1989.

[G1990] F. Glover, *Tabu search – Part II*, ORSA Journal on Computing, 2:4–32, 1990.

[G1995] González, Alejandro F., *Optimization Techniques in VLSI Phisical CAD*, EECS 527:CAD for VLSI, EECS Dept. Univ. of Michigan, Winter 1995, pp. 2-10.

[GD1996] L.M. Gambardella y M. Dorigo, *Solving symmetric and asymmetric TSPs by Ant colonies*, Proceedings of the IEEE Conference on Evolutionary Computation ICEC96 ,.IEEE Press 1996, pp. 622 –627.

[H] Talib S. Hussian, *Methods of Combining Neural Networks and Genetic Algorithms*, Queen's University.

[H1975] J.H. Holland, *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, 1975.

[H1986] P. Hansen, *The Steepest Ascent Mildest Descent Heuristic for Combinatorial Programming*, Presented at the Congress on Numerical Methods in Combinatorial Optimization, Capri, Italy 1986.

[H1992] Holland, John H., *Genetic Algorithms*, Scientific American, July 1992, pp 44-50.

[HPX] G. Harhalakis, J.M. Proth, X.L. Xie, *Manufacturing Cell Design Using Simulated Annealing: an Industrial Application*.

[HT1985] J. J. Hopfield y D. W. TANK, *'Neural' Computation of Decisions in Optimization Problems*, Biological Cybernetics 52, 1985, pp. 141–152.

[HW1990] A. Hertz y D. de Werra, *The Tabu Search Metaheuristic: how we used it*, Annals of Mathematics and Artificial Intelligence 1, 1990.

[JMM1996] Anil K. Jain, Jianchang Mao, K. Mohiuddin, *Artificial Neural Networks: A Tutorial*, IEEE Computer Special Issue on Neural Computing, March 1996.

[K1984] S. Kirkpatrick, *Optimization by simulated annealing: Quantitative studies*, Journal of Statistical Physics, 34:975–986, 1984.

[K1990] T. Kohonen, *The Self-Organizing Map*, Proceedings of the IEEE 78, 1990, pp. 1464–1480.

[KGV1983] S. Kirkpatrick, C. D. Gelatt Jr. y M.P. Vecchi, *Optimization by Simulated Annealing*, Science V. 220, No. 4598, 1983, pp. 671 – 680.

[MPP+1995] T. Mavridom, P.M. Pardalos, L. Pitsoulis, M.G.C. Resende, *Parallel Search for Combinatorial Optimization: Genetic Algorithms, Simulated Annealing, Tabu Search and GRASP*, Proceedings of the Workshop on Parallel Algorithms for Irregularly Structured Problems, Lyon, France, September 4-6 1995.

[PR2001] Petingi L. y Rodríguez J., *Reliability of Networks with Delay Constraints*, Submitted to Congressus Numerantium (2001).

[R1998] Mauricio G.C. Resende, *Greedy Randomized Adaptive Search Procedures (GRASP)*, AT&T Labs Research Technical Report, December 22 1998, 98.41.1.

[S1987] J.D. Schaffer, *Some Effects of Selection Procedures on Hyperplane Sampling by Genetic Algorithms*, Genetic Algorithms and Simulated Annealing, L. Davis ed. Pitman 1987.

[S1999] Smith, Kate A., *Neural Networks for Combinatorial Optimization: A Review of More Than a Decade of Research*, INFORMS Journal on Computing, Winter 1999, Vol 11, No 1.

[SD] Alice E. Smith y Berna Dengiz, *Evolutionary Methods for Design of Reliable Networks*, A chapter in Telecomunications Optimization: Heuristic and Adaptive Methods (D. Corne, editor), Wiley.

[SD1998] Alice E. Smith, Darren L. Deeter, *Economic Design of Reliable Networks*, IIE Transactions, Special Issue on Economics of Reliability Engineering, July 1998.

[SH1989] S. Sahni y E. Horowitz, *Fundamentals of Computer Algorithms*, Computer Science Press, 1989.

[SH1997a] T. Stützle y H. Hoos, *The MAX –MIN Ant system and local search for the traveling salesman problem*, Proceedings of IEEE-ICEC-EPS '97 IEEE International Conference on Evolutionary Computation and Evolutionary Programming Conference ,IEEE Press 1997, pp. 309 –314.

[SH1997b] T. Stützle y H. Hoos, *Improvements on the ant system:Introducing MAX –MIN ant system*, Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms, Springer Verlag, Wien 1997, pp. 245 –249.

[V1994] Valduriez, Patrick, *Algunas Ideas para mejorar la Escritura de Informes Técnicos*, Project Rodin INRIA, Rocquencourt, France, February 7 1994.

[W1993] Whitley, Darrell, *A Genetic Algorithm Tutorial,* Technical Report CS-93-103, Colorado State University, November 10 1993.

[Y1991] Yao Xin, *Optimization by Genetic Annealing*, Proceedings of the Second Australian Conference on Neural Networks ed. M.Jabri, Sydney-Australia, 1991, pp. 94-97

Capítulo 9
Apéndices

9. Apéndices

9.1.1. Mejores Prácticas para el Correcto Uso de los Algoritmos y Principales Objetos Proporcionados

En el curso de cualquier trabajo que pretenda reutilizar lo hecho por nosotros, ya sea para integrarlo sin modificaciones a un sistema mayor o para modificarlo (eventualmente ampliándolo), deberán tenerse en cuenta ciertos aspectos de bajo nivel, los que dependiendo de su naturaleza pueden volver engorrosa la tarea de integración, eventualmente haciendo que las estimaciones de tiempos para las tareas relacionadas indiquen valores mayores a los que realmente pueden alcanzarse, hecho que pude ocasionar una incorrecta toma de decisiones.

Dado que pretendemos que nuestro trabajo permita complementar, o incluso ser la base de trabajos futuros, dedicamos una sección entera a la explicación en detalle de los aspectos que, luego de conocidos, resultan clave para llevar a cabo tareas de integración o modificación en corto tiempo y de la mejor manera posible. Tareas críticas para el desarrollo de productos en los que la integración con nuestro trabajo resulte eficiente en el consumo de tiempo de procesador y espacio de memoria, que proporcione resultados confiables y que insuma un tiempo de implementación razonablemente corto.

En esta sección presentamos lo que consideramos constituye la forma correcta de uso, tanto de los objetos como de los algoritmos que proporcionamos en nuestra implementación.

El hacer invocaciones a nuestros algoritmos internamente desde código C++, los tiene las siguientes características destacables:

- No utiliza archivos para el intercambio de información entre la aplicación en cuestión y nuestro “motor de cálculo”.
- Permite a los desarrolladores de la aplicación en cuestión reutilizar código implementado por nosotros, tanto para utilizar nuestro “motor de cálculo” como para otros fines, los que pueden requerir o no la ampliación de las funcionalidades provistas por nosotros.
- Es posible acceder, finalizada la ejecución de nuestro “motor de cálculo”, a todas las soluciones encontradas que son equivalentes, en términos de intervalos de Diámetro-Confiabilidad, a la mejor solución descubierta en la búsqueda de la mejor solución, situación que puede darse dadas las limitaciones del método Monte Carlo Crudo, con el que para una solución dada no es posible obtener un valor concreto de Diámetro-Confiabilidad sino un intervalo de confiabilidades.

En alto nivel es posible decir que los pasos que recomendamos para realizar adecuadamente las invocaciones son los siguientes:

1- Colocar, dentro del código donde se pretende llevar a cabo la invocación, órdenes “include” que permitan incluir las componentes necesarias para la invocación de los algoritmos que hemos implementado.

2- En el código donde se va a llevar a cabo la invocación especificar lo siguiente:
   a. Crear un objeto de tipo grafo, utilizando el tipo de grafo adecuado según el algoritmo que se desee invocar, objeto al que llamaremos "objeto grafo" y al que denotaremos como "***ObjGrf***". En el caso de que se desee cargar el contenido de un archivo de grafos dentro del objeto ***ObjGrf*** puede invocarse al constructor de clases adecuado especificando el archivo de grafos desde el que se pretende cargar la información.
      i. En caso de la invocación al algoritmo de SIMPLIFICACIÓN pueden utilizarse los siguientes tipos de objeto:
         - *GrafoKDiamConfAmpliado*
         - *GrafoKDiamConfGRASP*
         - *GrafoKDiamConfGA*
      ii. En caso de la invocación al algoritmo GRASP se debe crear un objeto del tipo *GrafoKDiamConfGRASP.*
      iii. En caso de la invocación al algoritmo genético se debe crear un objeto del tipo *GrafoKDiamConfGA.*
   b. Cargar el objeto ***ObjGrf*** con los datos necesarios para representar el grafo con el que se quiere operar. Estos datos son los referentes a los nodos y las aristas del grafo en cuestión, y pueden colocarse dentro de la estructura de datos proporcionada por ***ObjGrf*** utilizando las operaciones para agregar nodos o aristas.
   c. Crear un objeto que permita almacenar el resultado devuelto por el algoritmo a invocar, llamaremos a este objeto "objeto resultado" y lo denotaremos con "***ObjRes***".
      i. En caso de la invocación al algoritmo de SIMPLIFICACIÓN este objeto deberá ser del tipo *GrafoKDiamConfAmpliado* o del mismo tipo que el "objeto grafo" creado en el paso a.
      ii. En caso de la invocación al algoritmo GRASP este objeto deberá ser del tipo *ResumenDesempenioGRASP*.
      iii. En caso de la invocación al algoritmo genético este objeto deberá ser del tipo *ResumenDesempenioGA*.
   d. Llevar a cabo la invocación del algoritmo adecuado para el objeto ***ObjGrf***, proporcionando los parámetros correspondientes y almacenando el resultado de la invocación en el objeto ***ObjRes***. La forma de hacerlo en el código fuente es utilizando una sentencia de la forma:

ObjRes=***ObjGrf****.AlgoritmoAdecuado(prémetros_necesarios)*

Finalizando la descripción de la mecánica a utilizar para la correcta invocación de nuestros algoritmos proporcionamos templates para los códigos de invocación. Debe tenerse en cuenta que dichos templates proporcionan ejemplos para clarificar la mecánica a utilizar, no deben tomarse como la única manera válida de llevar a cabo las invocaciones de los algoritmos SIMPLIFICACIÓN, GRASP ó Genético.

Nota: Por detalles de los parámetros que deben utilizarse y el significado de los mismos remitirse a los archivos de headers (.h) correspondientes.

Template para la invocación del algoritmo SIMPLIFICACIÓN

```
//Paso 1:
//Incluir el header necesario, puede ser cualquiera de los siguientes:
//   grafokdiamconfampliado.h
//   grafokdiamconfgrasp.h
//   grafokdiamconfga.h

include “grafokdiamconfampliado.h”
...

//Paso 2.a
GrafoKDiamConfAmpliado ObjGrf;
...

//Paso 2.b
  //Almacenar dentro de la estructura de datos del objeto ObjGrf los
  //datos del grafo con el que se pretende operar.
...

//Paso 2.c
GrafoKDiamConfAmpliado ObjRes;
...

//Paso 2.d
ObjRes=ObjGrf.Simplificación(parámetros adecuados);
  //ObjRes pasa a ser un grafo que es el resultado de simplificar el
  //grafo ObjGrf
...
```

Ejemplo para la invocación del algoritmo GRASP

```
//Paso 1:
include “grafokdiamconfgrasp.h”
...

//Paso 2.a
GrafoKDiamConfGRASP ObjGrf;
...

//Paso 2.b
  //Almacenar dentro de la estructura de datos del objeto ObjGrf los
  //datos del grafo con el que se pretende operar.
...


//Paso 2.c
ResumenDesempenioGRASP ObjRes;
...

//Paso 2.d
ObjRes=ObjGrf.GRASP(parámetros adecuados);
...
```

Ejemplo para la invocación del algoritmo GA

```
//Paso 1:
include “grafokdiamconfga.h”
...

//Paso 2.a
GrafoKDiamConfGA ObjGrf;
...

//Paso 2.b
  //Almacenar dentro de la estructura de datos del objeto ObjGrf los
  //datos del grafo con el que se pretende operar.
...

//Paso 2.c
ResumenDesempenioGA ObjRes;
...

//Paso 2.d
ObjRes=ObjGrf.GeneticAlgorithm(parámetros adecuados);
  //ObjRes pasa a contener el resultado de aplicar el algoritmo
  //genético sobre la red representada por el grafo ObjGrf,
  //utilizando además los parámetros quienes complementan la
  //definición de la instancia del problema a resolver.
...
```

9.2. Algoritmos Implementados

9.2.1. Código Fuente.

El código fuente desarrollado forma parte de la entrega en formato electrónico realizada en CD-ROM. Refiérase al archivo Leame.txt para determinar su ubicación.

9.3. Listado de Sugerencias Funcionales

El código fuente desarrollado forma parte de la entrega en formato electrónico realizada en CD-ROM. Refiérase al archivo Leame.txt para determinar su ubicación.

9.4. Recolección de Resultados

9.4.1. Formato de Archivos

El formato de los archivos esta dado a través de ejemplos, los que contienen una parte con valores para hacerlo más comprensible y otras con el formato genérico.

Formato Archivo de Pruebas - Compatible con Heidi (*.exp)

```
GRAFO
CONJUNTO DE NODOS
NODO
35            // identificador
1             // confiabilidad
0             // costo
0             // capacidad
FINNODO
NODO
...
FINNODO
...
FINCONJUNTO
CONJUNTO DE ARISTAS
ARISTA
14           // primer extremo
11           // segundo extremo
(14-11)      // identificador
66.054871    // costo
0.009089     // confiabilidad
1.317021     // capacidad
FINARISTA
ARISTA
...
FINARISTA
...
FINCONJUNTO
FINGRAFO
```

Formato Archivo de Ejecución de Pruebas – Portable a Windows/DOS (*.bat)
echo SIMPLIFICACION parámetros _ adecuados >>LogErrs.txt .\tallerv.exe SIMPLIFICACION 1 2 52.009386 00101.exp >>LogErrs.txt echo GRASP parámetros _ adecuados >>LogErrs.txt .\tallerv.exe GRASP parámetros _ adecuados >>LogErrs.txt echo GENETICO parámetros _ adecuados >>LogErrs.txt .\tallerv.exe GENETICO parámetros _ adecuados >>LogErrs.txt echo GRASP parámetros _ adecuados >>LogErrs.txt .\tallerv.exe GRASP parámetros _ adecuados >>LogErrs.txt echo GENETICO parámetros _ adecuados >>LogErrs.txt .\tallerv.exe GENETICO parámetros _ adecuados >>LogErrs.txt ... echo SIMPLIFICACION parámetros _ adecuados >>LogErrs.txt ...
- Las líneas que comienzan con “echo” agregan al log generado las invocaciones que fueron llevadas a cabo. - Las líneas que comienzan con .\tallerv.exe son las líneas de invocación al motor de cálculo, que dependiendo de su primer parámetro (SIMPLIFICACIÓN, GRASP o GA) aplica el algoritmo respectivo. Obsérvese que la salida de dichas ejecuciones es desviada al archivo de log, lo cual implementa el acumulado en el log de la información desplegada durante la ejecución.

Formato de Archivos de Resultado		
Algoritmo	**Nombre de archivo**	**Contenido del archivo**
Simplificación	*SPL*.TXT	NroNodos,NroAristas,NroNodosSimpl,NroAristasSimpl
GRASP	*GR*.TXT	IteraciónDeMuestreo,TiempoDeMuestreo,IteraciónMejorSolución,T oMejorSolución,MinIntervaloConfianza,MáxIntervaloConfianza,Re acionesMCC
GA	*GA*.TXT	GeneraciónDeMuestreo,TiempoDeMuestreo,GeneracionMejorSolución mpoMejorSolución,MinIntervaloConfianza,MáxIntervaloConfianza, icacionesMCC

Formato de Archivos de Consolidación – Compatible con MS Excel
Archivo consolidado por cada familia (RESUMEN*.CSV)
Grafo,Reporte,,GRASP,GRASP,GRASP,GRASP,GRASP,GRASP,GRASP,,GA,GA,GA,GA,GA,GA,GA Grafo,Reporte,,imuest,tmuest,imejor,tmejor,mininter,maxinter,repMCC,,gmuest,tmuest,gmejor,mininter,maxinter,repMCC
Archivo consolidado total (RESUMENESCONSOLIDADO.CSV)
MetricaFamilia,GRASP,GA,,#GRASP>#GA,#GA>#GRASP,#GRASP=#GA,,MáxGRASP,T.MáxGRASP,MáxGA,T.MáxGa

9.4.2. Juegos de Datos Utilizados

El juego de datos utilizado forma parte de la entrega en formato electrónico realizada en CD-ROM. Refiérase al archivo Leame.txt para determinar su ubicación.

9.4.3. Automatización de Consolidación de Resultados

Los scripts desarrollados durante la fase de pruebas para automatizar la recolección y consolidación de resultados forman parte de la entrega en formato electrónico realizada en CD-ROM. Refiérase al archivo Leame.txt para determinar su ubicación.

9.4.4. Archivos Resultados

Los archivos de resultados de las pruebas para los algoritmos de Simplificación, GRASP y Algoritmos Genéticos y aquellos que los consolidan forman parte de la entrega en formato electrónico realizada en CD-ROM. Refiérase al archivo Leame.txt para determinar su ubicación.