Algoritmos Genéticos Aplicados al Diseño de una Red de Comunicaciones Confiable

Daniel Calegari García
Proyecto de Grado
2002

Tutores
Hector Cancela
Sergio Nesmachnow

Instituto de Computación
Facultad de Ingeniería
Universidad de la República Oriental del Uruguay

RESUMEN

Durante el diseño de una red de comunicaciones, varias consideraciones deben ser tenidas en cuenta, dentro de las cuales se encuentran ciertas características de confiabilidad referentes a la topología de interconexión de sus nodos. Se podría decir que la confiabilidad global de una red es una medida que evalúa la probabilidad de éxito en la comunicación entre pares de nodos.

Tanto en la evaluación de la confiabilidad de estas redes como en el diseño optimo de las mismas, surge la necesidad de resolver distintos problemas, algunos de ellos con soluciones rápidas (como por ejemplo, cortes mínimos y flujos máximos), y otros NP-completos (como el calculo de árboles de Steiner de peso máximo). Para atacar los problemas de optimización NP-completos, una alternativa es el uso de metaheurísticas genéricas de optimización combinatoria (como las conocidas Algoritmos Genéticos, Simulated Annealing, etc), que intentan dar una alternativa flexible.

Este proyecto ataca el problema denominado Generalizado de Steiner (GSP) por medio de los Algoritmos Genéticos (AG) en su versión serial, para ello se proponen tres implementaciones del AG con diferentes representaciones del problema.

Se presentan diversos resultados experimentales derivados de la realización de pruebas de comparación entre las diferentes instancias del AG, y su comparación con resultados obtenidos por otras metodologías de resolución.

Palabras claves: Problemas NP-Completos, Algoritmos Genéticos, Simulated Annealing, Programación Evolutiva, Problema Generalizado de Steiner

Indice

Capítulo 1 Introducción

Durante el diseño de una red de comunicaciones, varias consideraciones deben ser tenidas en cuenta, dentro de las cuales se encuentran ciertas características de confiabilidad referentes a la topología de interconexión de sus nodos. Se podría decir que la confiabilidad global de una red es una medida que evalúa la probabilidad de éxito en la comunicación entre pares de nodos.

Tanto en la evaluación de la confiabilidad de estas redes como en el diseño optimo de las mismas, surge la necesidad de resolver distintos problemas, algunos de ellos con soluciones rápidas (como por ejemplo, cortes mínimos y flujos máximos), y otros NP-completos (como el calculo de árboles de Steiner de peso máximo). Para atacar los problemas de optimización NP-completos, una alternativa es el uso de metaheurísticas genéricas de optimización combinatoria (como las conocidas Algoritmos Genéticos, Simulated Annealing, etc), que intentan dar una alternativa flexible.

En un trabajo conjunto, el Departamento de Investigación Operativa y el Centro de Cálculo, han comenzado a investigar acerca de la resolución de problemas de optimización mediante técnicas de Programación Evolutiva, más precisamente Algoritmos Genéticos y su paralelización. Este proyecto intenta ser parte de ese trabajo conjunto, atacando el problema denominado Generalizado de Steiner por medio de los Algoritmos Genéticos en su versión serial.

En este capítulo introduciremos primero el problema concreto, una descripción de lo que son los Algoritmos Genéticos y por último, con una perspectiva más amplia, detallaremos los objetivos del proyecto.

1.1. Problema Generalizado de Steiner

El Problema Generalizado de Steiner (GSP), formulado por Krarup **[B1]** es un caso particular del llamado Problema de Steiner, el cual se trata con profundidad en el Apéndice 1 y es formulado de la siguiente manera:

Dado un grafo no dirigido $G=(V,E)$, una matriz de costos c asociados a las aristas, un subconjunto de nodos T denominados terminales, tal que $2 \le n_T \le n$ con $n=|V|$ y $n_T=|T|$, una matriz de dimensión $n_T \times n_T$, $R=\{r_{ij}\}_{i,j \in T}$ cuyos elementos son enteros positivos que indican los requerimientos de conectividad entre todo par de nodos de T. El resto de los nodos, son denominados de Steiner y pueden utilizarse o no en la solución dependiendo de la conveniencia de hacerlo.

Se pretende encontrar un subgrafo G_T de G de costo mínimo tal que para todo par de nodos $i, j \in T, i \neq j$, estos sean localmente r_{ij}-arista-conexos (deben existir r_{ij} caminos de aristas disjuntas).

Este será el problema a tratar, aunque existe una formulación análoga que implica caminos de nodos disjuntos.

Un ejemplo, así como su solución óptima se puede encontrar en la siguiente figura.

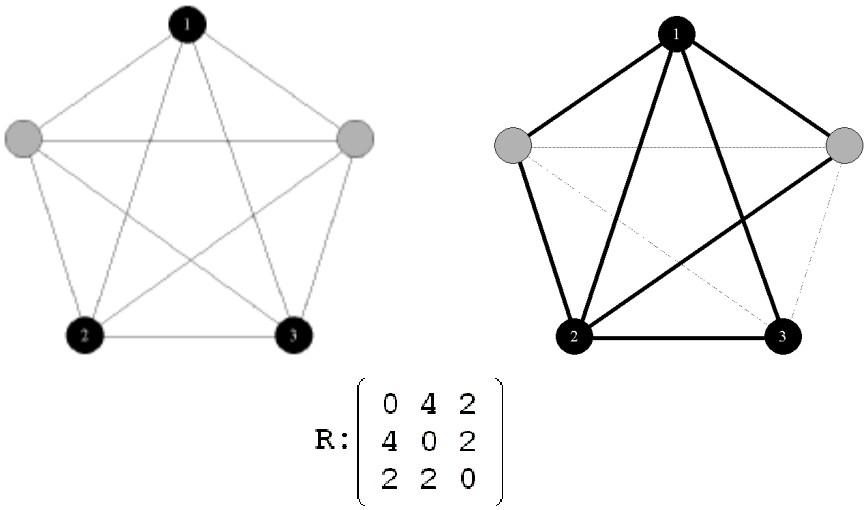


Figura 1.1 Problema y posible solución óptima al mismo.

A la izquierda podemos ver el grafo original. Los nodos negros numerados son los terminales, y los grises, los denominadosde Steiner. En este caso, el costo de las aristas está dado por el largo de estas. En el medio vemos la matriz de requerimientos entre los nodos terminales, por ejemplo: entre el nodo 1 y 2 deben existir 4 caminos. A la derecha podemos ver una de las posibles soluciones óptimas del problema.

1.2. Algoritmos Genéticos

La metaheurística denominada algoritmo genético (AG), se basa en un método iterativo para la resolución de problemas de búsqueda y optimización, principalmente. Un análisis más complejo se encuentra en el Apéndice 2.

El poder de esta técnica se basa en la robustez de la misma y que puede manejarse de forma exitosa en una gran gama de problemas, incluso en aquellos en los cuales se le hace difícil a otros métodos el hallar una solución. No es una técnica exacta, por lo que se buscan buenas aproximaciones. Su uso se fundamenta en base a la realización de una búsqueda eficiente en un espacio complejo y multidimensional.

Un AG es un algoritmo iterativo basados en imitar el proceso evolutivo de las especies. Básicamente este itera sobre un conjunto de datos (individuos), en cada iteración elige (selección) los mejores datos para generar (cruza y mutación) nuevos a usarse en la próxima iteración. El algoritmo termina en base a una función objetivo.

La estructura básica de un algoritmo genético es la siguiente:

```
begin
  pob := generar_población_inicial();
   /*Genero una población inicial de posibles soluciones
     al problema o aproximaciones a esta*/
  pob := calcular_fitness(Pob);
   /*Calculo el valor de aptitud de cada individuo*/

  while (not salir) do
    for 1 to largo(Pob) /2 do
      /*Tomo solo la mitad de los individuos para
        reproducir*/
      (ind1,ind2) := selecciono_dos_individuos(Pob);
      /*Selecciono dos individuos según su aptitud. Un
        mismo individuo puede seleccionarse dos veces*/
      (n_ind1,n_ind2) := cruza(ind1,ind2);
      /*Efectúo la "reproducción" generando dos nuevos
         individuos*/
      (n_ind1,n_ind2) := mutación(n_ind1,n_ind2);
      /*Efectúo la "mutación"*/
      (n_ind1,n_ind2) := calculo_aptitud(n_ind1,n_ind2);
      n_pob := n_pob + n_ind1 + n_ind2;
      /*La nueva población se forma a partir de los
        nuevos individuos.*/
    end for
    pob = n_pob;
    if (se cumple la condición de parada) then
      /*La condición de parada puede ser la convergencia
        del algoritmo a la solución esperada, la
        conclusión de un quantum de tiempo, etc*/
      salir = true;
    end if
  end while
end
```

Para definir un algoritmo genético debemos entonces definir los siguientes componentes:

- **Población de individuos:** cada uno de ellos representa una posible solución.
- **Función de aptitud de los individuos**: indica que tan bueno es el individuo en la población.
- **Función objetivo:** criterio de parada del algoritmo.
- **Selección y tasa de selección:** de que forma seleccionaremos los mejores individuos.
- **Cruzamiento y tasa de cruzamiento:** cómo combinaremos la información de dos individuos.
- **Mutación y tasa de mutación:** modificar parte de un individuo para introducir variedad en la población.

Quizás el punto más importante sea la definición de la estructura genética del individuo, o sea, de que forma se almacenará la información de una posible solución o aproximación. De esto dependerá luego la cruza y mutación para generar nuevos individuos y puede ser determinante para el algoritmo.

Estos conceptos serán mejor tratados en el capítulo 2 cuando se muestre la aplicación de esta metaheurística al problema particular.

1.3. Objetivos del Proyecto

Enmarcados en esta línea de trabajo, el proyecto propuesto consiste en el estudio del formalismo de algoritmos genéticos, y su implementación para la resolución del Problema de Steiner Generalizado, vinculado al diseño de redes de comunicaciones confiables.

El trabajo del proyecto propuesto comprende:

- Estudiar las propiedades del formalismo de algoritmos genéticos.
- Realizar un relevamiento de las herramientas de desarrollo de aplicaciones de algoritmos genéticos existentes.
- Introducirse a la temática de la confiabilidad de sistemas, y especialmente del problema mencionado anteriormente.
- Diseñar e implementar un algoritmo genético adecuado para dicho problema.
- Validar el trabajo realizado por medio de pruebas sobre diversas instancias concretas del problema.
- Comparar los resultados obtenidos entre diferentes implementaciones del algoritmo genético para hallar el mejor adaptado al problema.
- Comparar los resultados obtenidos con los resultados de otros métodos de optimización ya implementados en el grupo de trabajo o en otras instituciones.

En el capítulo 2 veremos de que forma se aplican los Algoritmos Genéticos a este problema, en el capítulo 3 trataremos la implementación de los algoritmos y la experimentación realizada en el capítulo 4. En el capítulo 5 presentaremos las conclusiones del proyecto y por último en los apéndices se presenta información extra de relevancia para el proyecto.

Capítulo 2 AG aplicados al Problema Generalizado de Steiner

No existe referencia de la utilización de Algoritmos Genéticos a la resolución del Problema Generalizado de Steiner, más que la de los realizados por esta línea de trabajo **[B21,B22]**.

Sin embargo, existe una amplia bibliografía de la aplicación de esta técnica para la resolución del Problema del Árbol del Steiner y el del Árbol Rectilíneo **[Apéndice 1]**, quizás por ser estos los de mayor aplicación.

A continuación presentaremos la descripción del algoritmo a implementar, así como un análisis de las estructuras genéticas a usar.

2.1 Algoritmo Genético

La estructura del algoritmo genético utilizado es la de uno simple, tal cual lo vimos cuando introdujimos los Algoritmos Genéticos.

Una de las decisiones a tomar es el criterio de parada que usará el AG. En principio se podría tomar un criterio de esfuerzo predefinido (parar por cantidad de generaciones), aunque esto implicaría la realización de un conjunto de pruebas de configuración a tales efectos que probablemente consuman mucho tiempo, y nada pueda extrapolarse sobre el valor de generaciones necesarias para diferentes dimensiones de problemas. Un segundo criterio a utilizar es el de medir la fluctuación de la calidad de las soluciones en la población, determinando la parada por un estancamiento de esta medida, luego de un determinado número de generaciones, lo suficiente para dar lugar a que la mutación tenga efecto sobre la población. Esto se tratará mejor en el próximo capítulo (Implementación).

Los parámetros a utilizar (lease: tamaño de la población, prob. de selección, tasa de mutación, tasa de cruza y cantidad de generaciones, en caso de usar una condición de parada de este tipo) serán determinados en base a un conjunto de pruebas de configuración, las cuales estás explicadas con mayor detalle en el capítulo 4 (Experimentación).

En particular existe un control realizado pre ejecución del algoritmo genético, éste realiza la prueba de factibilidad (tratada más adelante) del grafo original para determinar si éste puede llegar a resolver los requerimientos de conexión indicados o no. De esta manera se evita la realización del AG con requerimientos erróneos.

2.2 Representación Genética

La configuración del algoritmo genético requiere de la determinación de los parámetros de éste, como la tasa de cruza, selección y mutación, el largo de la población, etc.; y más importante aún la determinación de la estructura genética (genotipo) que tendrá un individuo para representar la solución, así como de los operadores (cruza, mutación) a aplicarse sobre esta.

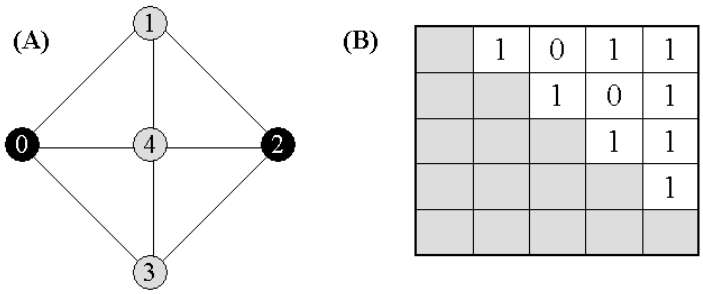
Gran parte del proyecto se centra en la puesta en práctica de tres estructuras definidas a continuación y su comparación tratada en el capítulo 4 (Experimentación), para determinar cual de estas logra un mejor desempeño en la resolución del problema generalizado de Steiner.

Un análisis completo de la implementación de estas representación puede verse en el capítulo 3 (Implementación).

2.2.1 Binaria

2.2.1.1 Representación

El genotipo es un arreglo booleano, el cual está compuesto por la unión de las filas de la matriz de adyacencia del grafo solución. Para realizar un almacenamiento más eficiente, debido a que el grafo original es no dirigido, se representa tan solo media matriz, la existencia de una arista es representada con el valor true en el arreglo, false sino.



(D) **Sea (i,j) la posición de la matriz y numV vértices, la posición del vector es:**

$$(i * numV) + j - \frac{(i+1)*(i+2)}{2}$$

Figura 2.1 Matriz (B) asociada al grafo (A) no dirigido. Representación binaria (C) de la información de la matriz y ubicación de cada gen (D)

2.2.1.2 Inicialización

La inicialización se realiza asignando valores aleatorios al arreglo verificando luego que éste represente una solución factible al problema de Steiner generalizado. En caso de no ser factible, se repite el proceso, hasta encontrar uno que sí lo sea.

La posible no factibilidad del individuo incurre en la pérdida de eficiencia del trabajo ya que es necesario, en tal caso, realizar el control de factibilidad y una posible corrección o descarte de este. La cruza y mutación de esta representación presentan el mismo problema.

2.2.1.3 Cruza

Se realiza la cruza simple de un punto como se puede ver en la figura 2. Dicha cruza puede generar genotipos inválidos, por lo que es necesario verificar la factibilidad de la solución.

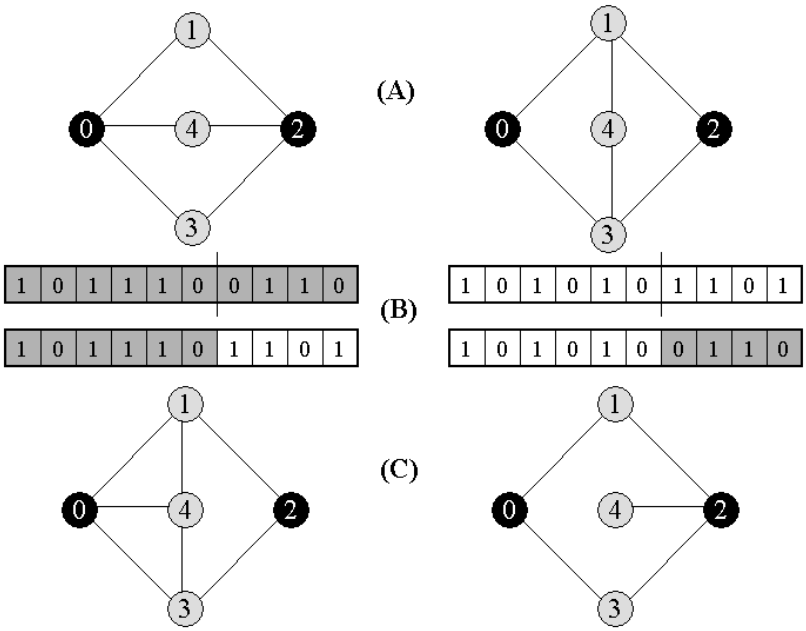


Figura 2.2 Soluciones iniciales (A) y su representación genética, cruza realizada (B) y genotipo final (C)

2.2.1.4 Mutación

Se decide aleatoriamente (con cierta probabilidad) si cada gen del genotipo debe mutarse o no. En este caso, se invierte el valor del gen, lo cual es equivalente a agregar o quitar una arista. Al igual que en la cruza, esto puede resultar en soluciones no factibles.

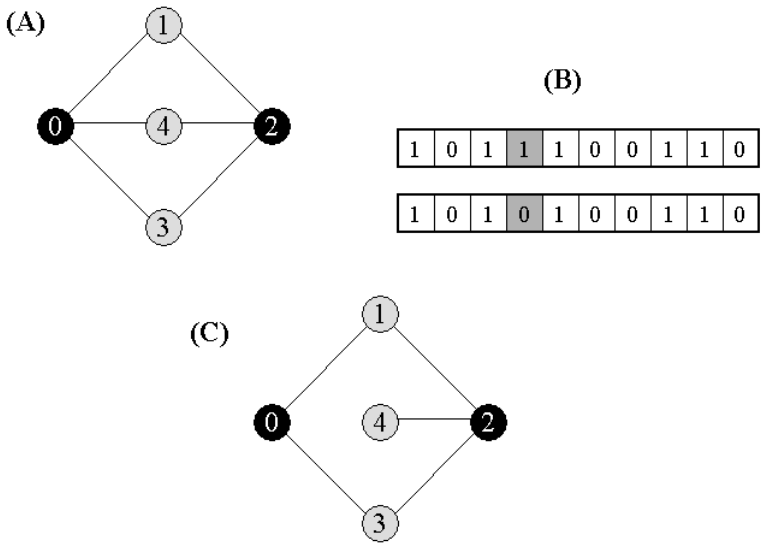


Figura 2.3 Mutación (B) del genotipo binario (A) y su resultado (C)

2.2.1.5 Factibilidad

Cómo se ha hecho explícito en las partes anteriores, esta representación no mantiene la factibilidad de la solución en los operadores genéticos ni en la creación de nuevos genotipos, por lo que es necesario de un algoritmo para tales efectos tratado al final del capítulo.

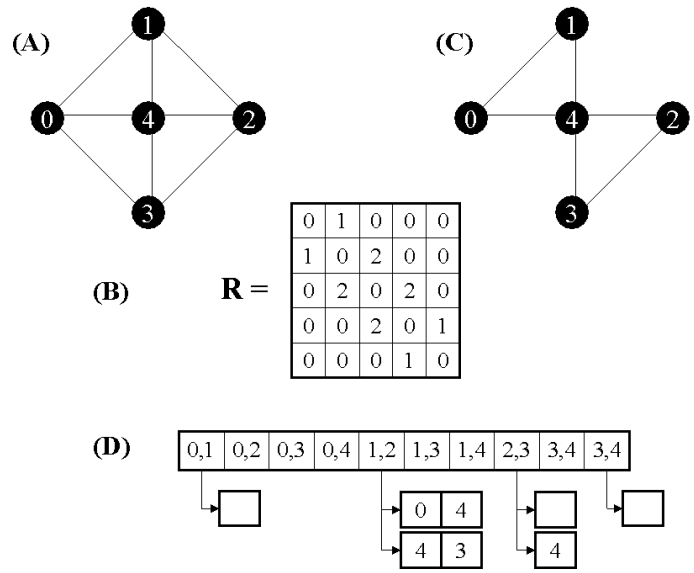
Esta representación tiene como ventajas, además de ser sumamente sencilla y eficiente en términos de memoria, no tener información replicada y permitir una simple realización de los operadores genéticos de cruza y mutación.

2.2.2 Lista de Caminos

2.2.2.1 Representación

Esta estructura representa una solución como un arreglo de largo igual a la cantidad de restricciones existentes. Cada gen contiene un conjunto de caminos disjuntos posibles entre los vértices de la restricción. Cada camino es especificado por una lista de identificadores de vértices por los cuales pasa el camino (por orden de aparición en el camino). Como se puede apreciar en la figura 2.2.4, solo se incluyen los identificadores de los vértices intermedios y una lista vacía si es un camino de largo 1.

Una ventaja de esta representación es que no se indican los vértices que no participan en la solución, sin embargo, si alguno está incluido en más de un camino, este se verá replicado.



0	1	0	0	0
1	0	2	0	0
0	2	0	2	0
0	0	2	0	1
0	0	0	1	0

Figura 2.4 Grafo original (A) y solución (C) posible según los requerimientos (B). Representación de listas de caminos de la solución (D). Existe redundancia de información.

2.2.2.2 Inicialización

Se genera inicialmente un grafo aleatorio que cumpla con las restricciones del GSP (igual que para la representación binaria) y con él, se realiza, para cada restricción, una búsqueda de los caminos necesarios para satisfacerla (deben existir tantos caminos disjuntos como indique la restricción). Esta búsqueda se realiza por medio del algoritmo de flujo máximo (Ford-Fulkerson, tratado al final del capítulo). Si bien este no devuelve uno a uno los caminos disjuntos, nos es indiferente ya que no nos interesa diferenciar por camino, sino obtener la solución por restricción. Obtendremos entonces caminos que entre ellos no son disjuntos pero la unión de todas las aristas genera la cantidad de caminos disjuntos deseada.

Una posibilidad, ya descartada, es la realización de un algoritmo de backtracking pero esto consumiría demasiado tiempo si se ejecuta en grafos de tamaño considerable.

2.2.2.3 Cruza

Se realiza también, una cruza de un punto, solo que en este caso la operación mantiene la solución factible ya que tan solo se cambian caminos por caminos manteniéndose las restricciones.

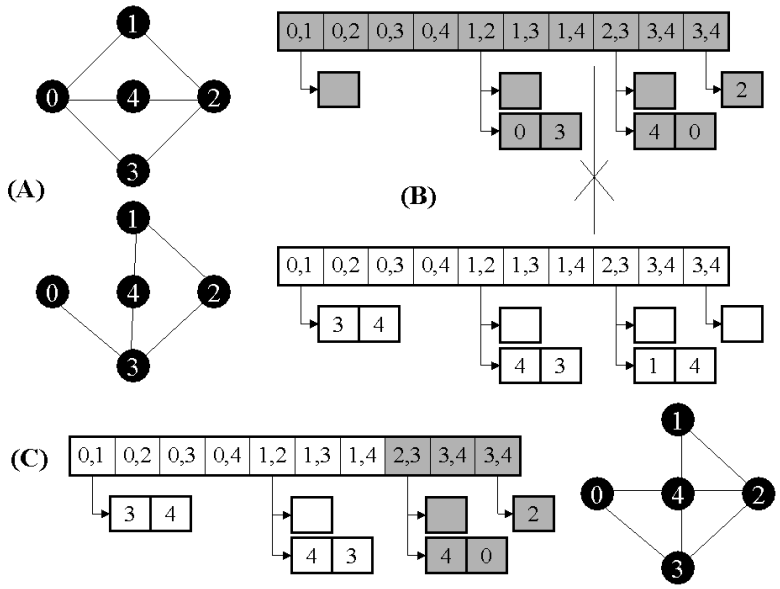


Figura 2.5 Soluciones iniciales (A) de los requerimientos de la figura 4 y su representación genética, cruza realizada (B) y uno de los genotipos finales (C). Mantiene la factibilidad.

2.2.2.4 Mutación

La mutación de un gen, implica volver a encontrar tantos caminos disjuntos nuevos como indique la restricción. Esta operación también mantiene la factibilidad de la solución. Previamente se seleccionan aleatoriamente (con cierta probabilidad), aquellos genes que deben ser mutados.

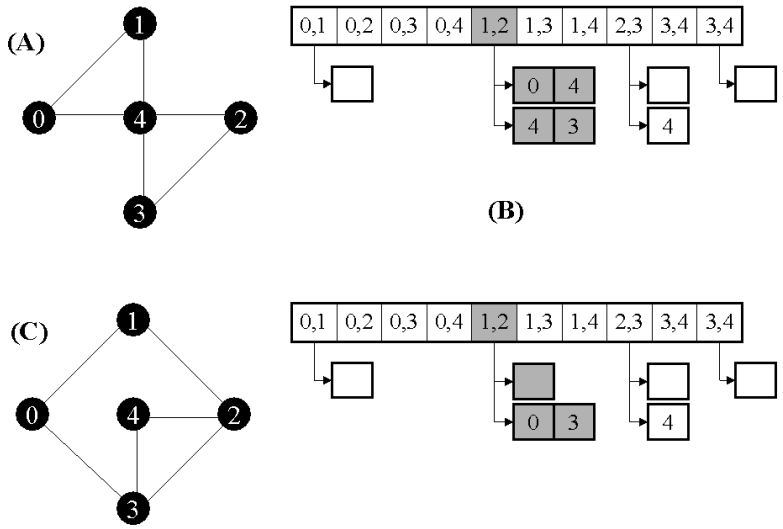


Figura 2.6 Mutación (B) del genotipo de listas de caminos (A) (requerimientos de la figura 4) y su resultado (C). Mantiene la factibilidad.

2.2.2.5 Factibilidad

Esta representación mantiene la factibilidad en la aplicación de los operadores genéticos.

La factibilidad continua que posee esta representación tiene como contrapartida la repetición del material genético

2.2.3 Subgrafo de Requerimientos

2.2.3.1 Representación

Esta representación es similar a la de lista de caminos, solamente que en cada restricción no representamos los caminos por separado, sino que almacenamos el grafo solución para ese requerimiento (Figura 2.2.7).

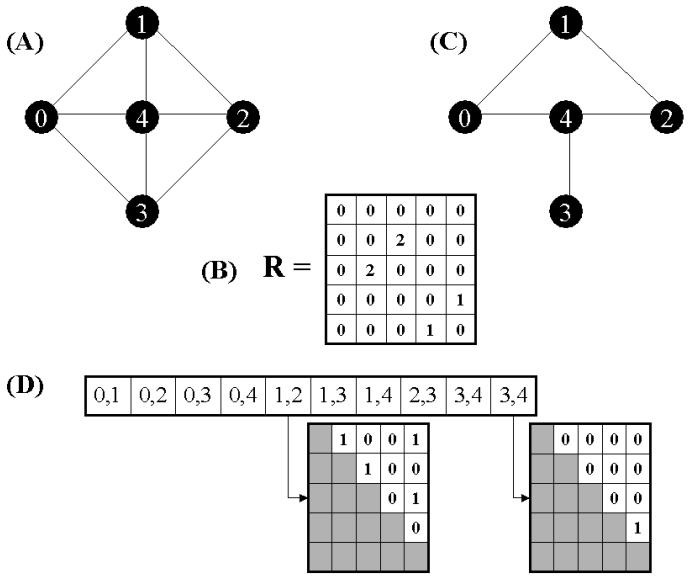


Figura 2.7 Grafo original (A) y solución (C) posible según los requerimientos (B). Representación de subgrafo de requerimientos de la solución (D). Existe redundancia de información.

2.2.3.2 Inicialización

La inicialización se realiza de igual forma que para la representación de lista de caminos, generando inicialmente un grafo aleatorio que cumpla con las restricciones del GSP, y hallando para cada restricción (utilizando el algoritmo de Ford-Fulkerson) el grafo que la satisfaga. La única diferencia entre ambas representaciones es que esta no duplica aristas por restricción.

2.2.3.3 Cruza

Se utiliza también una cruza de 1 punto.

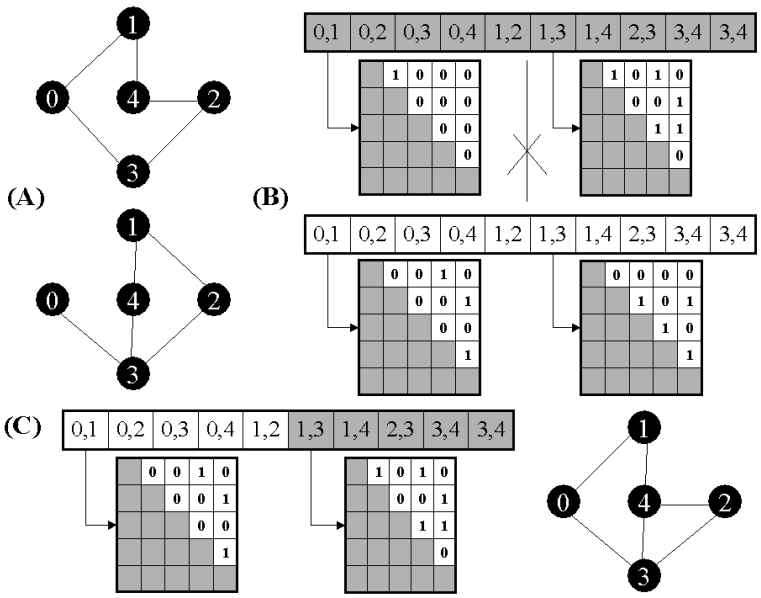


Figura 2.8 Soluciones iniciales (A) de los requerimientos de la figura 7 y su representación genética, cruza realizada (B) y uno de los genotipos finales (C). Mantiene la factibilidad.

2.2.3.4 Mutación

Al igual que en la representación anterior, se hallan nuevos caminos para una determinada restricción y se almacenan en un grafo solución.

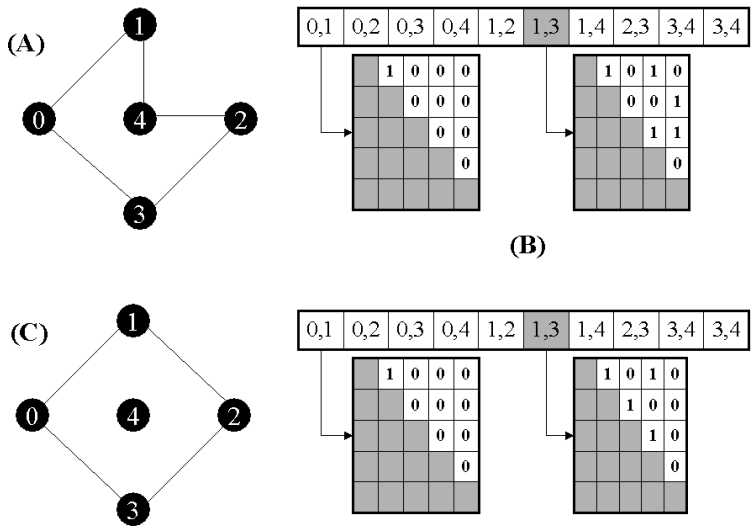


Figura 2.9 Mutación (B) del genotipo de listas de caminos (A) (requerimientos de la figura 7) y su resultado (C). Mantiene la factibilidad.

2.2.3.5 Factibilidad

Esta representación mantiene la factibilidad en la aplicación de los operadores genéticos y durante la creación de nuevos individuos.

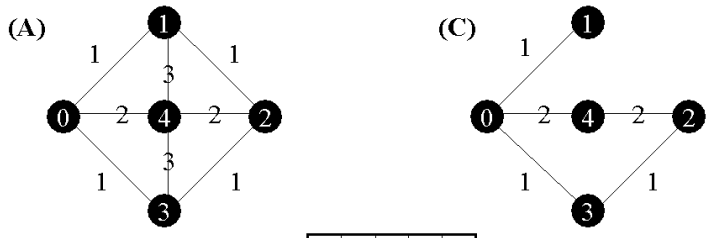
Básicamente esta representación es similar a la de listas de caminos no presentando diferencias aparentes.

2.2.4 Generalidades

2.2.4.1 Cálculo de Fitness

Contemplaremos la realización del cálculo del fitness de dos formas diferentes:

- **Diferencia:** es necesario, en todas las representaciones, calcular el costo total de la solución según el grafo original y restarle este al costo total del grafo original. Se suma uno al costo final para que ningún fitness sea cero (esto puede producir el mal funcionamiento del algoritmo genético).
- **Inversa:** es necesario, en todas las representaciones, calcular el costo total de la solución según el grafo original y devolver la inversa de este.



(B) $\mathbf{R} =$

0	1	0	0	0
1	0	0	1	0
0	0	0	1	0
0	1	1	0	2
0	0	0	2	0

(D)

$$Diferencia: CostoOriginal - NuevoCosto + 1 = 7$$

$$Inversa: \frac{1}{CostoOriginal} = \frac{1}{7}$$

Figura 2.10 Grafo original (A) y fitness (D) asociado a la solución (C) con requerimientos (B).

Durante la etapa de validación de algoritmo realizaremos un par de pruebas de forma tal de decidir cual de los cálculos del fitness es más apto para el problema.

2.2.4.2 Fitness Escalado

Goldberg [B12], introduce dos problemas relacionados con la utilización del fitness como factor determinante durante la selección. Estos implican una selección injusta cuando se trata de una población de pocos individuos, pudiendo ocurrir lo siguiente:

Indiferencia en la diversidad

Si los individuos de la población tienen una aptitud parecida, aunque representan soluciones diversas, no se logra distinguir entre soluciones con diferentes características, por lo que la selección se comporta indiferentemente entre los cromosomas.

Convergencia prematura

La existencia de pocos cromosomas de gran aptitud, genera el desplazamiento de los otros en pocas generaciones, generando una convergencia prematura.

La solución propuesta por Goldberg [B12] es la realización de un escalamiento del fitness. El escalamiento elegido para este trabajo es el lineal por su simplicidad.

Este define una función lineal F que depende del máximo fitness de la población, del fitness promedio y de un factor de escalado C, cuya ecuación es la siguiente:

$F(favg) = favg$
$F(fmax) = C * favg$

En donde: favg - fitness promedio de la población
fmax - fitness máximo de la población
C - factor de escalado

El factor C, es una medida del número de copias esperadas para el mejor individuo de la población (máximo fitness). Para pequeñas poblaciones (50 a 100 individuos), un factor de C de 1.2 a 2 puede ser usado satisfactoriamente.

La primera ecuación asegura que los cromosomas de mediana aptitud tengan una copia en las nuevas generaciones, mientras que la segunda ecuación regula el fitness de los cromosomas con máxima aptitud.

Un problema en el que incurre el escalado lineal es cuando es aplicado con valores muy cercanos a cero, pudiendo provocar valores de fitness negativo. Por esta razón no es posible utilizar el escalado lineal cuando el fitness se calcula con la inversa del costo de la solución. En ese caso se tomará el fitness en bruto (1/costo).

2.2.4.3 Cálculo de Factibilidad

Cualquiera que sea la representación usada, en algún momento es necesario realizar un cálculo de factibilidad sobre un individuo, que determinará si este continúa o no. Para tal acción es necesario verificar que la solución que representa es una solución válida de Problema Generalizado de Steiner.

Para ello, tal como se trata en [B22], primero se realiza el siguiente test heurístico que intenta controlar que el grado de los vértices terminales sea mayor o igual a la cantidad de caminos disjuntos que llegarán (saldrán) a ellos:

```
Para todo i dentro de vértices terminales
     Para todo j ≠ i dentro de vértices terminales.
          k = cantidad de caminos necesaria entre i y j.
          Si grado(i) < k o si grado(j) < k
               El grafo no es factible y salgo del test.
```

Esta heurística es claramente $O(|R|^2)$, por lo que es un test de bajo costo el cual puede llegar a descartar gran cantidad de casos.

En caso de que el test anterior sea realizado satisfactoriamente, es necesario determinar la existencia o no de cuanto camino disjunto deba existir, hallando, para cada par de vértices, la cantidad de caminos disjuntos existentes. Por tal motivo es utilizado el algoritmo de Ford-Fulkerson [B26] de la siguiente manera:

Para cada par de vértices terminales con requerimientos positivos, consideramos a la solución como una red, en la cual asignamos a los terminales el rol de fuente y pozo indistintamente. Cada arista se considera de dos sentidos (ya que el algoritmo trabaja sobre grafos dirigidos), cada una de costo unitario (indica que por allí se puede pasar solamente una vez).

La ejecución del algoritmo en un grafo de estas características indica el flujo máximo, el cual corresponde con el número de caminos disjuntos entre los vértices fuente y pozo (en este caso los terminales).

El orden de ejecución del algoritmo con estas características es de $O(|E| * r_{\max})$, siendo E el conjunto de aristas, y rmax el máximo $r_{ij} \in R$ [B26]. Como esto es realizado para cada par de vértices terminales, el orden de esta segunda parte es de $O(|E| * r_{\max} * |R|^2)$

Si la cantidad de caminos encontrada es menor que la necesaria, la solución que el individuo provee es considerada no factible y este es descartado.

Este algoritmo puede ser modificado para que devuelva las aristas que efectivamente participan en la solución. Esta modificación es utilizada para la creación de individuos en las representaciones de lista de caminos y la de subgrafo de requerimientos.

Capítulo 3 Diseño e Implementación del Algoritmo Genético

Si nos remontamos a la etapa de planificación del proyecto, podemos descubrir que la implementación del algoritmo genético proviene originalmente de un motor genérico que ha estado siendo desarrollado por integrantes del Centro de Cálculo. En un principio se propuso la idea de utilizar este motor genético para la implementación, pero esta opción fue descartada debido a que este se hallaba en una primera versión de prueba, sin documentación asociada y que no permitía tener un completo control sin la necesidad de realizar cambios que resultaran de gran impacto, pensando en la etapa de experimentación.

Por tales razones se optó por realizar una versión reducida del motor (sin toda la generalidad que este posee), optimizado para el problema en particular, basándose fuertemente en el diseño del original y manteniendo el mismo lenguaje de implementación (C sobre plataforma UNIX/Linux), de forma tal tener un total control y conocimiento de la aplicación, generando de esta forma cierto valor agregado al proyecto.

Se presentaran las características generales del motor y a su vez se tratará lo relativo a la implementación del GSP.

3.1 Estructura General

3.1.1 Algoritmo

Se implementó un algoritmo genético simple cuya estructura puede verse en la siguiente figura:

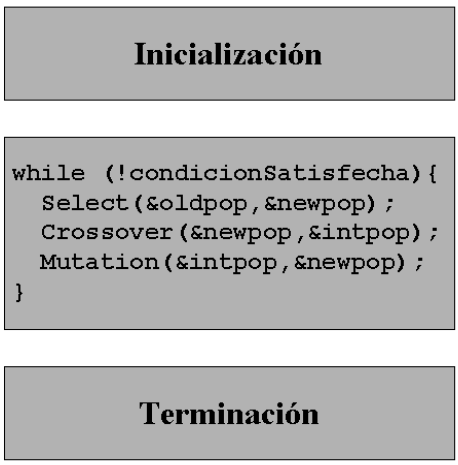


Figura 3.1 Estructura principal del AG.

Una primer etapa se encarga de la inicialización de la población, de la carga de los datos de configuración y de las creación de las utilidades auxiliares para la ejecución del algoritmo. Lo sigue el algoritmo propiamente dicho; fijarse que la iteración delega la realización de la selección, cruza y mutación a un operador genético poblacional y no realiza allí mismo los operadores primitivos a nivel de genotipo como introducimos en el capítulo 1 (esto no tiene ninguna incidencia más que introducir cierta modularidad). Por último tenemos una etapa de terminación, la cual imprime los resultados y estadísticas de la ejecución del algoritmo así como procede a la destrucción de las estructuras (población y utilidades) creadas en la primer etapa.

El siguiente es un diagrama de dependencias de código fuente (UML,**[B29]**), el cual muestra los diferentes componentes que integran la solución y sus dependencias. Cada uno de los componentes serán tratados con mayor profundidad a continuación.

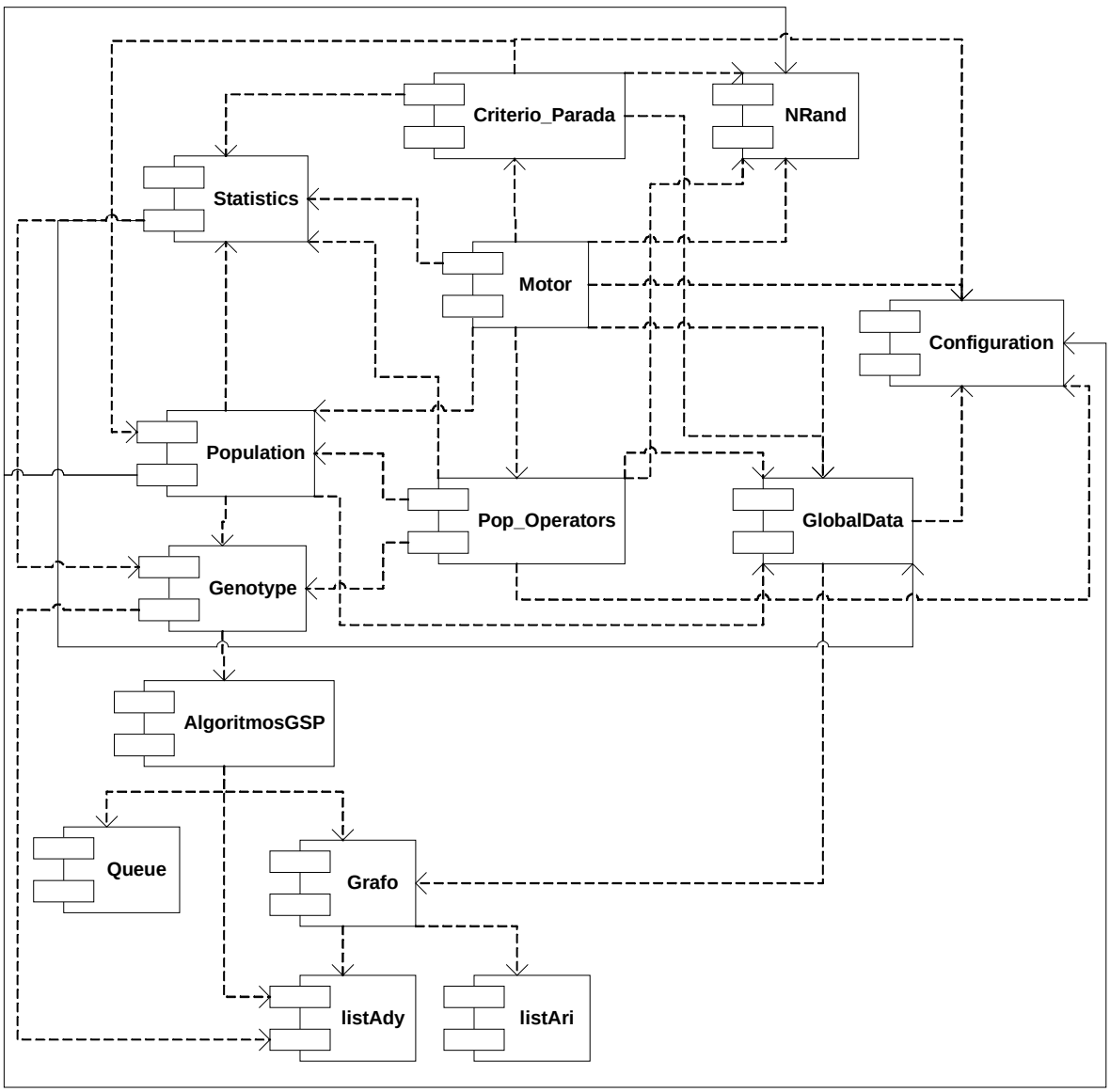


Figura 3.2 Diagrama de dependencias del código fuente.

3.1.2 Población y Operadores Genéticos

La población es simplemente un contenedor de genotipos con las operaciones triviales de obtener y ubicar estos dentro de la colección.

Además la población almacena información acerca de los valores máximo y mínimo del fitness con el objetivo de permitir realizar el escalado lineal de la población de manera más eficiente, teniendo ya calculados los valores de los factores a y b (ver sección Fitness Escalado en el Cápitulo 2).

```
typedef struct {
       Genotype *pop;
       int size;
       int capacity;
       float fitness;

       float min_fitness;
       float max_fitness;
       float fitness_esca;
       float a,b;
} Population;
```

Figura 3.3 Estructura de la población

Los operadores genéticos poblacionales (selección cruza y mutación) están encargados de seleccionar genotipos e invocar las operaciones de cruza y mutación a nivel de genotipo (no son más que controladores algorítmicos de operaciones más primitivas).

La estructura general del algoritmo no realiza ningún chequeo de factibilidad de los genotipos generados debido a que muchas veces esto no se utiliza, por tal razón se delega esa responsabilidad a las operaciones básicas de cruza y mutación a nivel de genotipo.

3.1.2.1 Selección

```
code Select (Population *old, Population *new, Nrand*, GlobalData*,
             Statistics*, Configuration*);
```

Se realiza la selección por ruleta utilizando los valores escalados del fitness. Es bueno aclarar que en caso de utilizarse un cálculo de fitness inverso, no se escala la población ya que lleva a la generación de valores de fitness negativos.

3.1.2.2 Cruza

```
code Crossover (Population *old, Population *new, Nrand*, GlobalData*,
                Statistics*, Configuration *);
```

Se itera eligiendo aleatoriamente dos genotipos y se realiza el cruzamiento entre ellos. El algoritmo sigue hasta que se realizan satisfactoriamente la cantidad de cruzamientos necesarios para conformar la nueva población.

3.1.2.3 Mutación

```
code Mutation (Population *old, Population *new, Nrand*, GlobalData*,
               Statistics*, Configuration*);
```

A cada genotipo de la población se le indica que debe mutarse. Esta mutación siempre es válida (devuelve siempre un genotipo válido aunque este sea el mismo).

3.1.3 Genotipo

La definición del genotipo es realizada por el usuario ya que depende estrictamente del problema a resolver y de la estructura que queramos utilizar para tal resolución.

El genotipo posee además de operaciones simples del estilo de la creadora, destructor, calcular fitness, etc, operaciones para realizar la cruza y mutación de estos. Estas dos operaciones son particulares de cada genotipo pero en líneas generales deben devolver genotipos válidos y realizar el chequeo de factibilidad en caso de que sea necesario.

3.1.3.1 Representaciones

Los genotipos asociados con el problema GSP son los siguientes:

La representación binaria es la más simple, se representa como un arreglo booleano representativo de la parte superior de la matriz de adyacencia que representa la solución al problema.

```
typedef struct {
       boolean *rep;
       int numV;
       int size;
       float fitness;
  } Genotype;
```

Figura 3.4 Representación Binaria

```
typedef struct {
       int terminal_1;
       int terminal_2;
       int cantCam;
       listAdy **caminos;
  } nodoReq;

typedef struct {
       nodoReq *rep;
       int cantReq;
       float fitness;
  } Genotype;
```

Figura 3.5 Representación de Lista de Caminos

La representación de lista de caminos presenta una complejidad superior a la anterior ya que es un arreglo de estructuras que representan la solución de una restricción. Esta solución está representada por un arreglo de listas de enteros, cada uno de los cuales identifican un nodo.

La representación de subgrafo de requerimientos es similar a la de lista de caminos con la diferencia que en vez de tener un arreglo de listas, se tiene un grafo que representa al requerimiento.

```
typedef struct {
       int terminal_1;
       int terminal_2;
       int cantCam;
       Grafo *caminos;
  } nodoReq;

typedef struct {
       nodoReq *rep;
       int cantReq;
       float fitness;
  } Genotype;
```

Figura 3.6 Representación de Subgrafo de Requerimientos

3.1.3.2 Estructuras y Algoritmos Asociados

Las representaciones de los genotipos utilizan dos estructuras auxiliares un Grafo con estructura de listas de adyacencia y una lista de enteros denominada lista de adyacencia (utilizada además en la estructura interna del Grafo).

Tanto para el chequeo de factibilidad como para la creación de los genotipos se utiliza un conjunto de algoritmos que actúan sobre un grafo, definidos en AlgoritmosGSP.

```
int FactibleGSP (Grafo *grafo, Grafo *req, int *terminales, int cantTer);
```

Indica si el primer grafo cumple con los requerimientos de conectividad (indicados en req) del problema GSP entre todo par de nodos terminales (indicados en terminales). Es utilizado para los chequeos de factibilidad, debiendo los genotipos representar su solución como un grafo a tales efectos.

```
int CantidadCaminos (Grafo *grafo, int Vsal, int Vlle, int numC);
```

Indica si en el grafo existen o no numC caminos disjuntos entre los nodos referenciados por sus números (Vsal y Vlle).

```
code HallarCaminos(listAdy *lista, Grafo *grafo, int Vsal, int Vlle,
                   int numC, Nrand *generator);
```

Devuelve un arreglo de lista de adyacentes que significan numC caminos disjuntos entre los nodos Vsal y Vlle en el grafo indicado. Es una precondición que existan por lo menos numC caminos. El algoritmo se basa en el de flujo máximo (Ford-Fulkerson), por lo que los caminos no tienen por que ser disjuntos entre sí, pero la unión de todos los caminos satisfacen el requerimiento. Se optó por este algoritmo ya que halla soluciones rápidamente en cualquier tipo de problema, las otras opciones (backtracking, DFS aleatorio) pueden llegar a consumir un tiempo demasiado grande dependiendo del problema particular. Esta operación es usada para la creación de genotipos de la representación de Lista de Caminos.

```
code HallarCaminosGrafo(Grafo *newg, Grafo *grafo, int Vsal, int Vlle,
                        int numC, Nrand *generator);
```

Es una operación idéntica a la anterior, solo que en este caso se devuelve un grafo representativo de la solución y no una lista de caminos. Esta operación es usada para la creación de genotipos de la representación de Subgrafo de Requerimientos.

3.1.4 Aleatoriedad

Es necesario que el algoritmo genético sea no determinista, para ello se utiliza un generador de números aleatorios basado en una secuencia congruencial lineal tratada en **[B30]**:

Xn+1 = (a*Xn + c) mod m
tiene periodo **m** sii
i) **c** es relativamente primo a **m**
ii) **a-1** multiplo de **p**, para todo **p** factor de **m**
iii) **a-1** multiplo de 4 si **m** multiplo de 4

Figura 3.7 Secuencia congruencial lineal

El algoritmo se inicializa en base a dos semillas de valores aleatorios y posee procedimientos que permiten la obtención de valores aleatorios enteros, reales, booleanos, uniformes y entre cierto rango.

3.2 Utilidades

El motor provee de un conjunto de utilidades las cuales posibilitan la adaptabilidad de este a otro tipo de problemas así como la configuración de sus parámetros de forma dinámica y la recolección de datos durante su ejecución.

3.2.1 Configuración

Este componente permite la lectura de parámetros desde un archivo de texto, el cual es especificado en la invocación del algoritmo genético.

La invocación del algoritmo genético será:

```
motor[Binario|LisCam|SubReq]  -c  <configuration_file>
```

Estos parámetros son los parámetros genéticos, las direcciones de los archivos de salida de la ejecución, la dirección del archivo con la semilla de los números aleatorios y cualquier otro dato que sea necesario para los genotipos, datos globales, etc.

Los parámetros son accedidos por medio del nombre que los identifica y su valor asociado es devuelto por operaciones a tales efectos.

El formato general del archivo se puede ver en el **Apéndice 4: Ejecución del Algoritmo**.

3.2.2 Datos Globales

Los datos globales representan datos que deberán ser accedidos constantemente y por eficiencia se almacenan juntos y pasados por referencia a cada operación que los necesite.

Dependiendo del problema se puede definir nuevos datos globales, lo que lleva a tener que recompilar el motor pero sin deber realizar modificaciones sobre el resto, más que por aquellos componentes definidos por el usuario que utilicen estos datos (se trata con más profundidad en la siguiente sección).

En particular, para el GSP se utilizan como datos globales el grafo que representa el GSP, el grafo de requerimientos y una lista de los nodos terminales. Estos datos son necesarios para el cálculo de factibilidad de los genotipos.

```
typedef struct {
      Grafo red;
      Grafo reqs;
      int *terminales;
      int cantTerminales;
  } GlobalData;
```

Figura 3.8 Datos Globales asociados al GSP.

3.2.3 Criterio de Parada

En vista que queríamos tener varias opciones que nos permitieran determinar el criterio de parada del algoritmo, se pensó en realizar una estructura que almacena información del estado general del algoritmo y puede determinar cuando este debe detenerse.

Para poder manejar los dos criterios de interés (cantidad de iteraciones y calidad de solución) se definió la siguiente estructura:

```
typedef struct {
       int generaciones;
       int actual;
  } crit_gen;

typedef struct {
       int numGenTot, numGen;
  } crit_cal;
```

```
typedef struct {
       union {
         crit_gen *criterio1;
         crit_cal *criterio2;
       } criterio;

       Genotype max_fitness;
       int tipo_criterio;
  } CriterioParada;
```

Figura 3.9 Estructura del Criterio de Parada

La estructura se compone de una unión de criterios, a los cuales se les puede sumar otros. En general se mantiene una copia del genotipo de máximo fitness en la población con el objetivo de no perder la información de este a medida que avanza el algoritmo.

El criterio de cantidad de iteraciones almacena un contador de iteraciones (actual), el cual se va incrementando, definiendo la parada cuando alcanza el valor de generaciones.

Por su parte el criterio de calidad de solución comienza a chequearse cuando el fitness máximo de la población es menor que el que CriterioParada posee. Ahí se comienza a contar el número de iteraciones (numGen) y si ese número alcanza numGenTot (definido en base a la cantidad de generaciones que debe haber hasta que la mutación se realice) se indica la terminación del algoritmo. En caso de que el fitness sea mayor, se vuelven los valores a cero.

La decisión de la parada del algoritmo se toma en base a la invocación de la operación:

```
int condicionSatisfecha (CriterioParada*, Population*, Statistics*,
                         Configuration*);
```

que como se puede observar no solamente obtiene los datos de la condición de parada sino también del estado del algoritmo determinado por la población actual.

3.2.4 Estadística

Para la correcta realización de la etapa de experimentación (Capítulo 4) es necesario obtener ciertos datos de la ejecución del algoritmo. Por esta razón se definió una estructura denominada **Statistics**, la cual almacena información que se va generando a lo largo de la ejecución.

Dentro de los datos almacenados se encuentran dimensiones (bytes) de individuos, tiempos de duración del algoritmo y de las operaciones más importantes, datos del mejor genotipo e información de cada una de las generaciones.

La recolección de información se realiza pasando una referencia de esta estructura a toda operación de interés (operadores genéticos, creadoras, etc)

```
typedef struct {
       long int tiempo, generacion;
       float best_fit, fit_prom_pob;
       struct IterationData *next;
  } IterationData;

typedef struct {
       long int by_pob, vel_cr_pob;
       float by_prom, t_cr_prom, t_mut_prom, vel_cr_prom;
       long int t_cr_tot, t_mut_tot;
       long int cant_gen, t_ejecucion, t_evol;
       Genotype *best_genotype;
       long int iter_best_gen;
       IterationData *data_iter;
  } Statistics;
```

Figura 3.10 Estructura de estadísticas

Capítulo 4 Experimentación

La experimentación se divide en tres etapas, primeramente realizamos una etapa de validación del algoritmo, la cual intenta detectar errores del algoritmo y proveer una versión final de este. Seguidamente realizamos una etapa de configuración, en la cual determinamos los parámetros más aptos para la ejecución del algoritmo. Por último se realizan las pruebas de comparación, las cuales tienen como objetivo la realización de las conclusiones finales.

Uno de los aspectos más importantes de esta fase es lograr recolectar un "buen" conjunto de datos de prueba, donde la noción de bueno depende de varios factores entre ellos la heterogeneidad del mismo y la posible utilización previa de estos problemas junto con su resolución en base a diferentes metodologías. Por esta razón se ha utilizado para las pruebas problemas de **[B1]** y **[B22]**, los cuales cumplen con las características antes mencionadas, incluidos estos en el CD adjunto.

4.1 Validación

Las pruebas de validación son realizadas para verificar el correcto funcionamiento del sistema para un conjunto de pruebas particulares. El conjunto de prueba es el utilizado en **[B1]** y se basa en topologías de pequeña dimensión con soluciones óptimas o aproximadas conocidas con el agregado de dos grafos de mediana (Problema 8) y gran dimensión (Problema 9), provenientes de **[B1]** y **[B22]** respectivamente. Todos estos grafos se encuentran en el CD adjunto.

Se deben validar ciertos aspectos a continuación detallados:

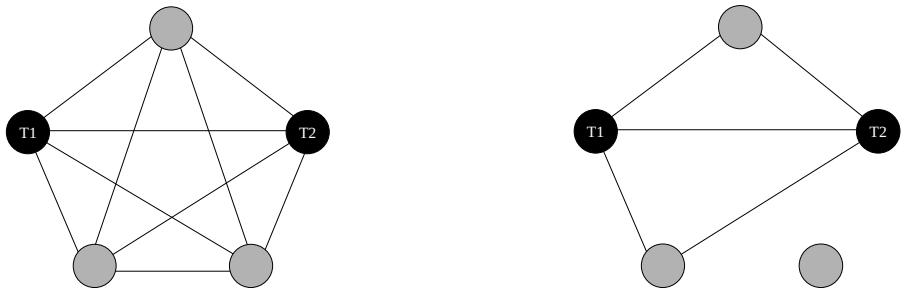
- **Correcto funcionamiento del algoritmo:** para esto debemos verificar que funcione correctamente la generación de individuos, los criterios de parada, los operadores genéticos y los diferentes cálculo de fitness.
- **Robustez:** realizamos las pruebas con problemas de diferente porte, incrementando su dimensión y verificamos que sea factible la utilización del algoritmo en términos de memoria y tiempo de ejecución.

Esta verificación debe ser llevada a cabo para cada una de las estructuras definidas en el capítulo 2.

4.1.1 Casos de Prueba

En la representación gráfica de los grafos asociados a las instancias de GSP elegidas como casos de prueba, los nodos grises representan los nodos del tipo Steiner, mientras que los nodos negros representan los nodos terminales. Las aristas están etiquetadas con sus costos y los requerimientos de conexión están dados por la matriz R.

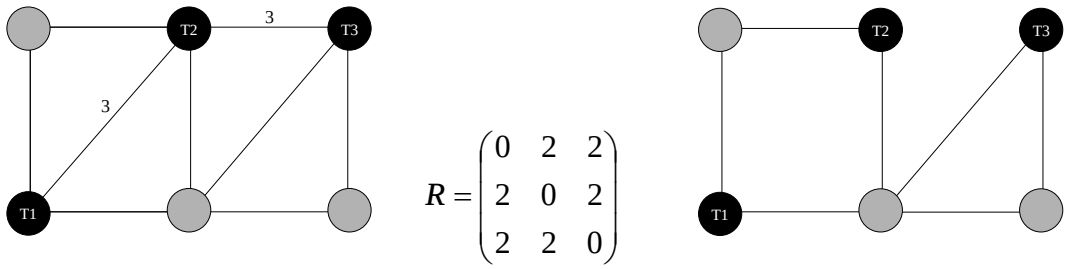
4.1.1.1 Problema 1 (PV1)



$$R = \{r_{ij} = 3, \forall i, j \in T\}$$

Figura 4.1 Problema propuesto (izquierda) donde las aristas tienen costo 1, requerimientos de conexión (centro) y posible solución óptima (derecha) de costo 5. Grafo 1 (K5) extraído de [B22].

4.1.1.2 Problema 2 (PV2)



$$R = \begin{pmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{pmatrix}$$

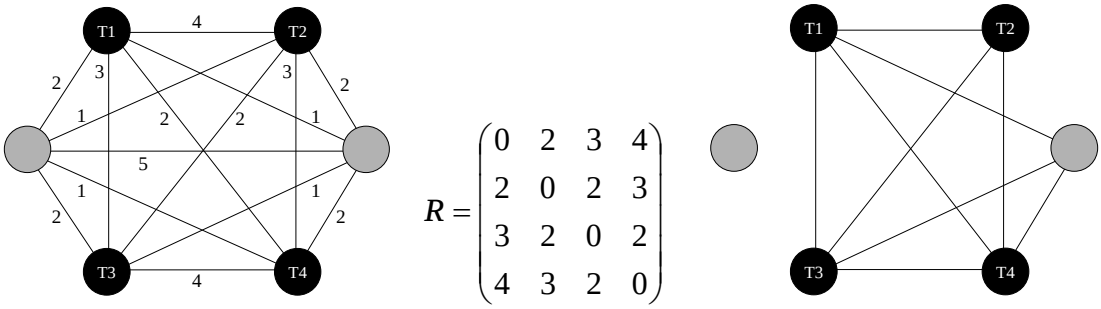
Figura 4.2 Problema con 2 aristas de costo 3 y el resto de costo 1 (izquierda). Requerimientos de conexión (centro) y posible solución óptima de costo 7 (derecha). Grafo 2 extraído de [B22].

4.1.1.3 Problema 3 (PV3)

$$R = \{r_{ij} = 3, \forall i, j \in T\}$$

Figura 4.3 Problema con aristas de costo 1 (izquierda), requerimientos de conexión y posible solución óptima de costo 9 (derecha). Grafo 3 extraído de [B22].

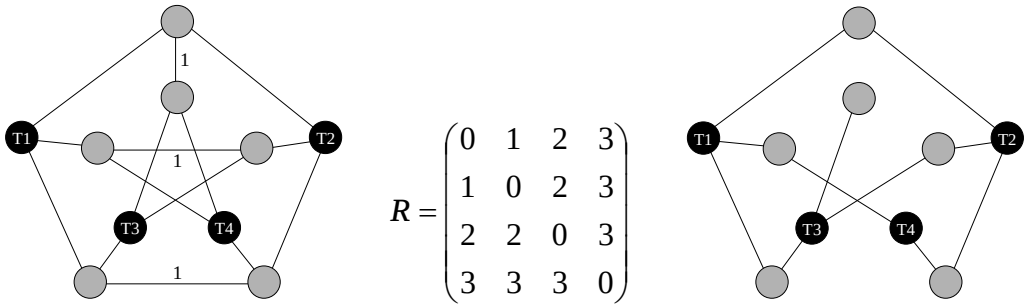
4.1.1.4 Problema 4 (PV4)



$$R = \begin{pmatrix} 0 & 2 & 3 & 4 \\ 2 & 0 & 2 & 3 \\ 3 & 2 & 0 & 2 \\ 4 & 3 & 2 & 0 \end{pmatrix}$$

Figura 4.4 Problema planteado (izquierda), requerimientos de conexión (centro) y solución aproximada (izquierda) de costo 22. Grafo 4 (K6) extraído de [B22].

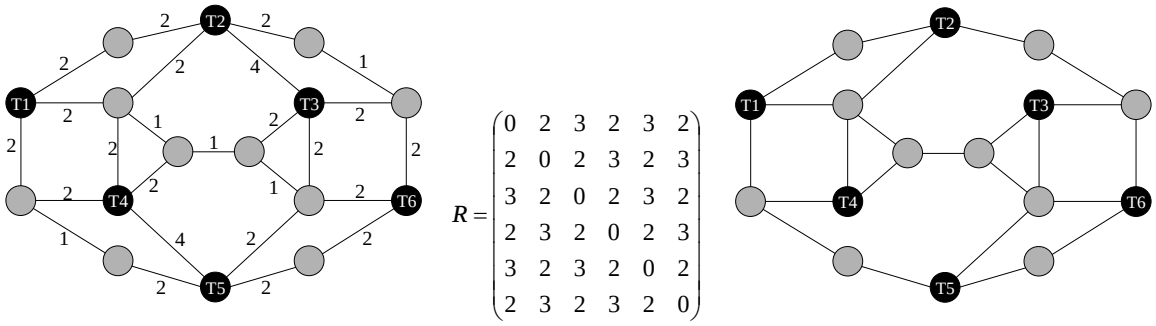
4.1.1.5 Problema 5 (PV5)



$$R = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 2 & 3 \\ 2 & 2 & 0 & 3 \\ 3 & 3 & 3 & 0 \end{pmatrix}$$

Figura 4.5 Problema donde el resto de las aristas tienen costo 2 (izquierda), requerimientos de conexión (centro) y solución óptima de costo 22 (izquierda). Grafo 5 (Peterson) extraído de [B22].

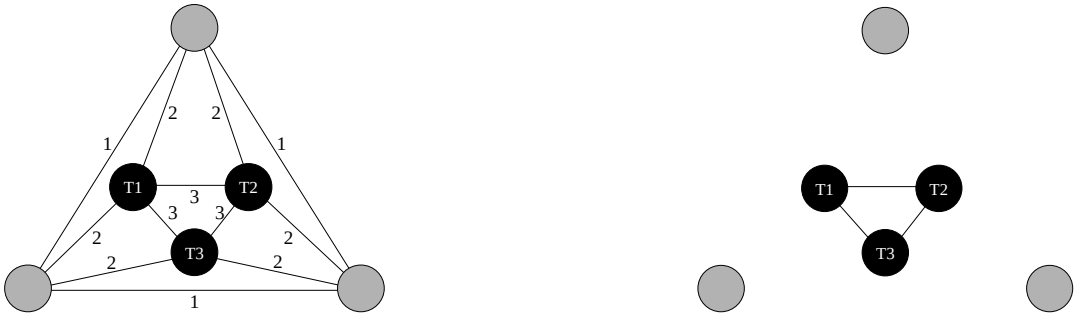
4.1.1.6 Problema 6 (PV6)



$$R = \begin{pmatrix} 0 & 2 & 3 & 2 & 3 & 2 \\ 2 & 0 & 2 & 3 & 2 & 3 \\ 3 & 2 & 0 & 2 & 3 & 2 \\ 2 & 3 & 2 & 0 & 2 & 3 \\ 3 & 2 & 3 & 2 & 0 & 2 \\ 2 & 3 & 2 & 3 & 2 & 0 \end{pmatrix}$$

Figura 4.6 Problema planteado (izquierda), requerimientos de conexión (centro) y solución aproximada de costo 41 (derecha). Grafo 6 extraído de [B22].

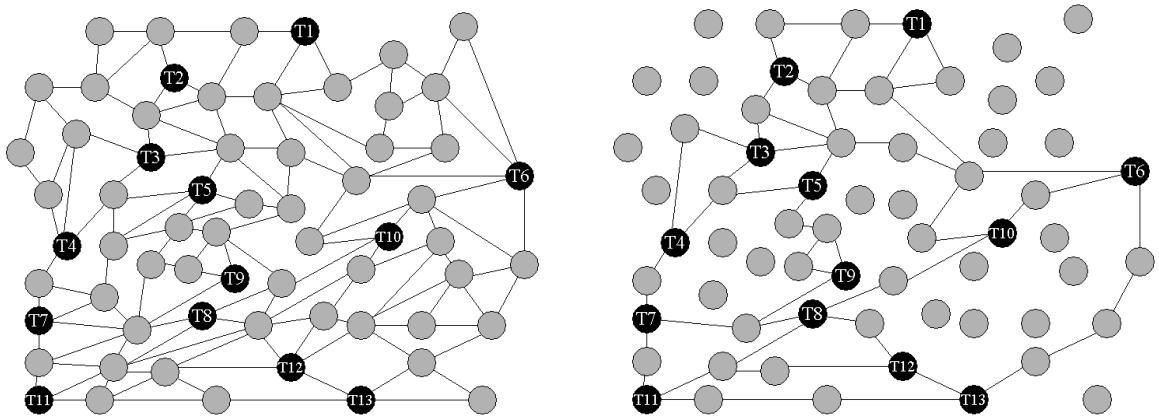
4.1.1.7 Problema 7 (PV7)



$$R = \{r_{ij} = 2, \forall i, j \in T\}$$

Figura 4.7 Problema (izquierda), requerimientos de conexión (centro) y solución óptima de costo 9 (derecha). Grafo 7 extraído de [B22].

4.1.1.8 Problema 8 (PV8)



$$R = \begin{pmatrix} 0 & 2 & 1 & 3 & 2 & 2 & 3 & 2 & 1 & 2 & 2 & 2 & 2 \\ 2 & 0 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 3 & 2 & 2 & 2 \\ 1 & 1 & 0 & 2 & 2 & 2 & 2 & 3 & 3 & 1 & 1 & 2 & 2 \\ 3 & 1 & 2 & 0 & 2 & 2 & 2 & 2 & 2 & 1 & 2 & 2 & 1 \\ 2 & 2 & 2 & 2 & 0 & 1 & 1 & 1 & 2 & 2 & 1 & 2 & 2 \\ 2 & 2 & 2 & 2 & 1 & 0 & 2 & 2 & 3 & 3 & 2 & 1 & 2 \\ 3 & 2 & 2 & 2 & 1 & 2 & 0 & 1 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 3 & 2 & 1 & 2 & 1 & 0 & 3 & 2 & 2 & 2 & 2 \\ 1 & 2 & 3 & 2 & 2 & 3 & 2 & 3 & 0 & 1 & 1 & 1 & 1 \\ 2 & 3 & 1 & 1 & 2 & 3 & 2 & 2 & 1 & 0 & 2 & 2 & 2 \\ 2 & 2 & 1 & 2 & 1 & 2 & 2 & 2 & 1 & 2 & 0 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 1 & 2 & 2 & 1 & 2 & 3 & 0 & 2 \\ 2 & 2 & 2 & 1 & 2 & 2 & 2 & 2 & 1 & 2 & 3 & 2 & 0 \end{pmatrix}$$

Figura 4.8 Problema (izquierda) con 51 nodos del tipo Steiner, 13 nodos terminales y una alta densidad de aristas. Los enlaces punto a punto entre nodos del tipo Steiner tienen costo 1, un enlace entre un nodo del tipo Steiner y un nodo terminal tiene costo 2 y un enlace entre dos nodos terminales tiene costo 4. La matriz de requerimientos de conexión tiene diferentes tipos de requerimientos para distintos pares de nodos terminales (centro). La solución óptima (derecha) tiene costo 99. Grafo genérico denso extraído de [B22].

4.1.1.9 Problema 9 (PV9)

Grafo de gran dimensión con 100 vértices, gran densidad de aristas (400 aprox.) y una amplia gama de requerimientos de conexión (10 terminales). Este grafo fue generado aleatoriamente para las pruebas realizadas en **[B22]** denominado grafo_2_0 y forma parte de las pruebas de configuración (Problema 5) y comparación (Problema 1).

4.1.2 Realización de las pruebas

Para una correcta validación, es necesario validar todos los componentes de la aplicación. Algunos de ellos, como las estructuras y los operadores genéticos, se validan mediante la ejecución del algoritmo y la observación de la solución a la que llegan. Otros tales como el criterio de parada y el tipo de cálculo del fitness necesitan de un análisis más profundo ya que las pruebas no solo son necesarias para decidir su correcto funcionamiento sino también para optar dentro de las diferentes opciones. Los criterios de parada que tenemos como opción son: la calidad de la solución o un esfuerzo predefinido por medio de la cantidad de iteraciones, la cual dependerá del tamaño del problema. Dentro de las opciones de cálculo del fitness tenemos: realizar la diferencia del costo del grafo original y de la solución o la inversa del costo del grafo solución. Hay que recordar que en este último caso no es posible escalar el fitness por lo ya explicado en el Capítulo 3 (Cálculo del Fitness).

Por tal razón se deben validar las diferentes combinaciones realizando para cada problema y representación, una ejecución de la aplicación por combinación, verificando si llega a una solución (puede no hacerlo por poseer memoria insuficiente o haberlo detenido por consumir demasiado tiempo) y en caso afirmativo: el costo de la misma, el tiempo de ejecución, el número de iteraciones y la iteración en la que se encuentra el mejor genotipo, lo cual nos ayudará a definir en la etapa de configuración la combinación más apta. La representación gráfica de los resultados se puede observar en la siguiente tabla:

		Criterio de Parada	
		Calidad de Solución	**Esfuerzo Predefinido**
Fitness	**Inverso**	**(Fin?,Costo,Tiempo,#Iter,#IterBest)**	**……**
	Diferencia	**……**	**……**

Figura 4.9 Tabla de resultados. Se tendrá una por problema.

En vista que aún no se han realizado las pruebas de configuración, se deben tomar valores de parámetros posibles, aunque no sean los mejores. Estos valores pueden modificarse en caso de que sea necesario, ya que puede no observarse el correcto funcionamiento con los valores predeterminados. Los parámetros por defecto elegidos fueron los siguientes:

- **Número de individuos en la población:** 50
- **Tasa de mutación:** 0.003
- **Tasa de cruzamiento:** 0.9
- **Tasa de selección:** 0.6
- **Factor de escalado:** 2

Las pruebas de validación fueron realizadas en una máquina con procesador **Pentium IV de 512 Mb de memoria RAM y 1.6 GHz de reloj**.

Los resultados obtenidos pueden apreciarse en el Apéndice 3. y se incluyen en el CD adjunto

Debemos realizar dos observaciones en los datos obtenidos: la correctitud de la aplicación y la robustez de la misma ante casos de prueba de dimensiones considerables.

4.1.2.1 Funcionamiento del Algoritmo

Los problemas más pequeños muestran que el algoritmo halla los mejores resultados o soluciones aproximadas para cualquiera de las tres representaciones en muy poco tiempo; Un ejemplo de esto son los resultados para el problema 7 cuyo óptimo posee un costo de 9.

Binaria	**(1,9,375,100,56)**	**(Fit. Inverso, Esfuerzo Pred.)**
Lista de Caminos	**(1,9,109,13,1)**	**(Fit. Inverso, Cal. De Sol.)**
Subgrafo de Requerimientos	**(1,9,148,15,1)**	**(Fit. Diferencia, Cal. De Sol.)**

Figura 4.10 Mejores resultados para el problema 7 (Pv7), se indica a la derecha, la combinación con la que se halló cada resultado (cálculo de fitness y criterio de parada).

En general vemos que el algoritmo evoluciona correctamente para cualquiera de las combinaciones de cálculo de fitness y criterio de parada. Estos problemas, sin embargo, son de una dimensión demasiado pequeña para realizar alguna otra consideración al respecto por lo que han sido utilizados para dar un primer paso en la validación del algoritmo.

Por otro lado los problemas de mayor dimensión (problemas 8 y 9), nos dan una verdadera medida del correcto funcionamiento del algoritmo. Cabe recordar que para el problema 8 el óptimo tiene costo 99 y para el problema 9 no se conoce un óptimo, pero los mejores costos hallados rondan en el entorno de los 400.

Consideremos para la validación dos casos distintos: primero la validación de las estructuras complejas, las cuales conviene analizarlas juntas ya que son y segundo la validación de la representación binaria.

Como dijimos, las representaciones complejas tienen una gran similitud, tanto en estructura como en comportamiento (la forma que generan los individuos, los operadores genéticos, etc) y esto se ve reflejado en los resultados. Analizando los datos generacionales, podemos ver la correcta evolución del algoritmo para ambas representaciones aún en el peor caso, sin embargo podemos ver en muchos casos que el algoritmo comienza con individuos muy buenos y le dificulta evolucionar, muchas veces se estanca en buenas soluciones iniciales y la mutación no surte efecto, quizás porque esta no posibilita eliminar aristas si estas están repetidas en más de una restricción. Un ejemplo de esto es la evolución del algoritmo para el problema 8 en el peor caso de la representación de Lista de Caminos (Figura 4.11).

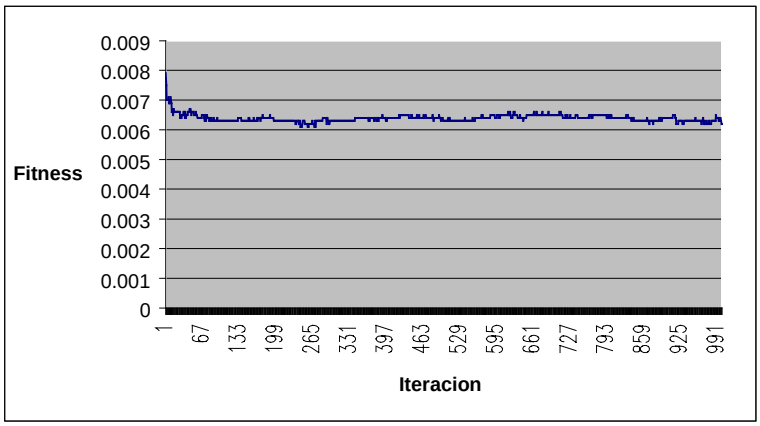


Figura 4.11 Evolución del fitness máximo para la representación de Lista de Caminos sobre el problema 8 en el peor caso.

Vemos además que ambas están cerca de óptimo y para los mejores casos se obtienen sub óptimos no muy lejanos al óptimo global. Los mejores resultados para este problema son para cada combinación (cálculo de fitness y criterio de parada):

Lista de Caminos	**(1,122,91606,13,1)**	**(Fit. Diferencia, Cal. De Sol.)**
Subgrafo de Requerimientos	**(1,122,98788,13,1)**	**(Fit. Diferencia, Cal. De Sol.)**

La realidad mostrada por la representación binaria es similar para los problemas del 1 al 8, en los cuales se puede apreciar la evolución del algoritmo y una buena convergencia a los mejores valores. Un ejemplo de esta buena convergencia es la detección del mejor valor para el problema 8 (fitness óptimo es 83), lo cual se ve en la figura siguiente.

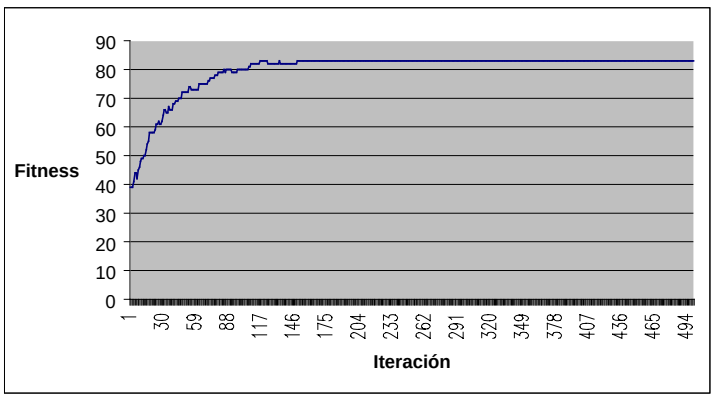
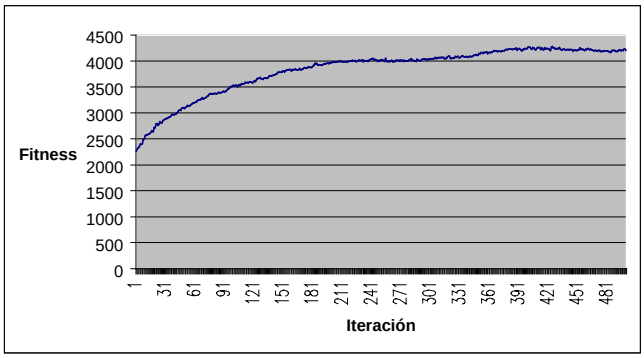


Figura 4.12 Evolución del mejor fitness del mejor caso para la representación binaria sobre el problema 8 en la cual halla la solución óptima.

Sin embargo, se puede apreciar que para el problema 9, ésta representación presenta resultados (Figura 4.13) lejos del mejor valor conocido (costo de 350 aprox.). Esto se puede deber a la lenta evolución que posee la población, la mala elección de la semilla de números aleatorios, etc, por lo que no podemos afirmar que resulte obsoleta ya que sí podemos apreciar la evolución de esta, más aún teniendo en cuenta que uno de los resultados puede ser considerado aceptable (Costo 646 hallado en la iteración 425 de 500).

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,2889,272930,100,86)	(1,2012,1308075,500,490)
	Diferencia	(1,1499,328820,124,110)	(1,646,1201924,500,425)

Figura 4.13 Resultados de la representación binaria aplicada al problema 9.



Binaria	(1,646,1201924,500,425)
Lista de Caminos	(1,470,2042254,1000,939)
Subgrafo de Requerimientos	(1,470,2462300,1000,939)

Figura 4.14 Evolución del mejor fitness para la representación binaria en el problema 9 para el mejor caso y comparación con los mejores resultados de las otras estructuras.

Existen algunas otras apreciaciones que pueden ser realizadas acerca de los resultados obtenidos, las cuales sirven como base para posteriores etapas de experimentación:

Eficiencia

Esta medida la consideramos en términos de utilización de memoria y tiempo necesario por iteración. Tal como podemos apreciar en la figura 4.15, la representación binaria es la que más memoria consume para el almacenamiento de un individuo. Esto se debe a que la misma representa todas las posibles aristas del grafo, aunque estas no existan, mientras que las demás representaciones solo consideran las aristas existentes. El segundo lugar es ocupado por la representación de Lista de Caminos. La diferencia entre esta y la de Subgrafo de Requerimientos es que esta contiene aristas repetidas mientras que la otra no.

El tiempo que consume la creación de individuos es similar en las tres representaciones ya que todas comienzan a partir de un grafo aleatorio hallado de igual manera, la diferencia mínima entre la representación Binaria y las otras es que esta se queda con ese grafo aleatorio mientras que para las otras es necesario hallar soluciones individuales por restricción.

En cuanto al tiempo necesario para efectuar la cruza, podemos ver que la representación Binaria consume un tiempo muy superior, esto es debido a que es la única que necesita el chequeo de factibilidad. Sin embargo, es la que consume menor tiempo a la hora de realizar la mutación debido esto a que las representaciones complejas necesitan hallar una nueva solución para una determinada restricción. Estas diferencias hacen que el tiempo de ejecución promedio sea parecido para las tres representaciones (en 500 iteraciones, el tiempo de ejecución de las representaciones complejas se reduce aproximadamente a la mitad en los resultados de la figura 4.15).

Mejores Resultados

Es difícil indicar qué representación provee los mejores resultados en esta instancia, ya que no hemos realizado suficientes pruebas para ello. Sin embargo cabe notar, como se ha dicho antes, que la representación binaria ha mostrado una convergencia lenta, además de que las representaciones complejas comienzan con mejores soluciones, debido esto a que es más pulida la creación de individuos. En general todas han logrado hallar buenos resultados.

Mejor Combinación

Hay que considerar para la etapa de configuración que hasta el momento la combinación de tipo de cálculo de fitness y criterio de parada que mejores resultados ha obtenido en los grafos grandes ha sido la de un cálculo de fitness por diferencia de costos y la de un criterio de parada por cantidad de iteraciones.

PV9	Binaria	Lista de Caminos	Subgrafo de Requerimientos
Bytes promedio de individuo	19812 (~15Kb)	1825.12 (~1.5Kb)	660 (~0.5Kb)
Bytes total de Población	990600 (~900Kb)	91256 (~90Kb)	33000 (~30Kb)
Tiempo promedio de creación (ms)	42.2800	57.0200	61.1400
Tiempo total de creación (ms)	2114 (~2 seg)	2851 (~2.8 seg)	3057 (~3 seg)
Tiempo de cruza promedio (ms)	936.0699	12.8901	154.4945
Tiempo de cruza total (ms)	468971	12903	154649
Tiempo de mutación promedio (ms)	1454.7146	2019.3457	2269.4126
Tiempo de mutación total (ms)	728812	2021365	2271682
Tiempo total de ejecución (ms)	1201924 (~20 min)	2042254 (~34 min)	2462300 (~41 min)
Cantidad de generaciones	500	1000	1000
Mejor individuo (iteración,costo)	(425,4280)	(939,4456)	(939,4456)

Figura 4.15 Información general de la ejecución del algoritmo en el mejor caso para el problema 9. Las poblaciones fueron en los tres casos de 50 individuos, el tipo de cálculo de fitness fue la diferencia y el criterio de parada fue por cantidad de iteraciones (500 en la binaria, 1000 en las otras).

4.1.2.2 Robustez

En cuanto a la robustez, queda mostrado que todas las representaciones poseen un comportamiento aceptable para problemas de mediano-gran porte como el problema 9.

Un cálculo realizado a “grosso modo” nos muestra que la cantidad de memoria necesaria por individuo es de aproximadamente de 20 Kb para la representación binaria, 2 Kb para la de Lista de Caminos y 1 Kb para la de Subgrafo de Requerimientos.

4.2 Configuración

Durante las pruebas de validación hemos utilizado valores de los parámetros genéticos por defecto, aunque no tenemos la certeza de que sean los mejores.

Por tal razón se realizan las pruebas de configuración, las cuales tienen como objetivo principal determinar los valores más aptos (aunque sea un tanto subjetivo) para la ejecución del algoritmo, en base a un conjunto de pruebas heterogéneo extraido de **[B1]** y **[B22]** (Problema 5), de los parámetros:

- **Cantidad de generaciones (si se utiliza este criterio de parada)**
- **Tamaño de la población**
- **Tasa de mutación**
- **Tasa de cruzamiento**
- **Tasa de selección**
- **Factor de escalado**

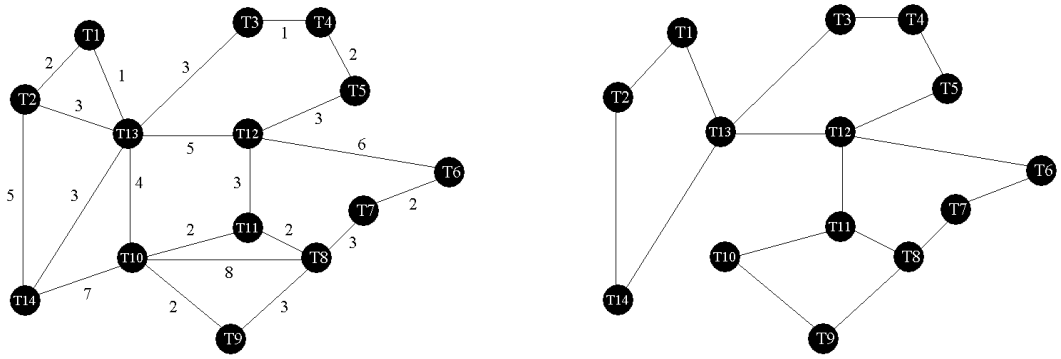
Un objetivo previo de la configuración de parámetros, es tomar una decisión acerca de los criterios de parada y los tipos de cálculo del fitness a usarse en la aplicación.

Los grafos utilizados en las pruebas se pueden encontrar en el CD adjunto.

4.2.1 Casos de Prueba

En la representación gráfica de los grafos asociados a las instancias de GSP elegidas como casos de prueba, los nodos grises representan los nodos del tipo Steiner, mientras que los nodos negros representan los nodos terminales. Las aristas están etiquetadas con sus costos y los requerimientos de conexión están dados por la matriz R.

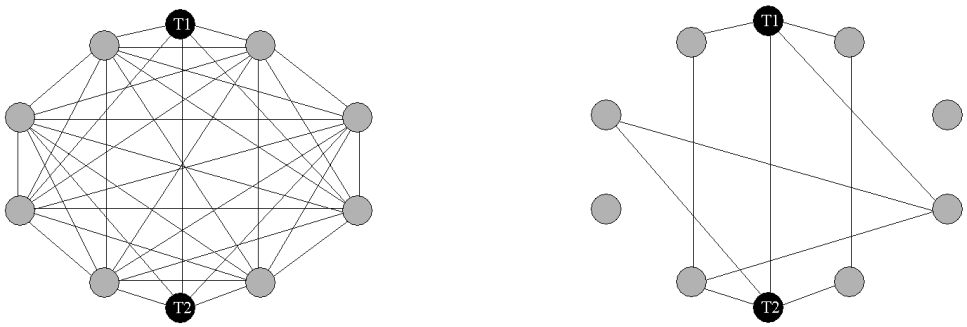
4.2.1.1 Problema 1 (PCf1)



$$R = \{r_{ij} = 2, \forall i, j \in T\}$$

Figura 4.16 Red 2-arista-conexa tomando como base el núcleo de fibra óptica de la red telefónica de Montevideo (izquierda). Se asumen costos heterogéneos de conexión entre los diferentes nodos terminales. Requerimientos de conexión (centro) y solución óptima del problema de cost 46 (derecha). Red telefónica de Montevideo extraído de [B1].

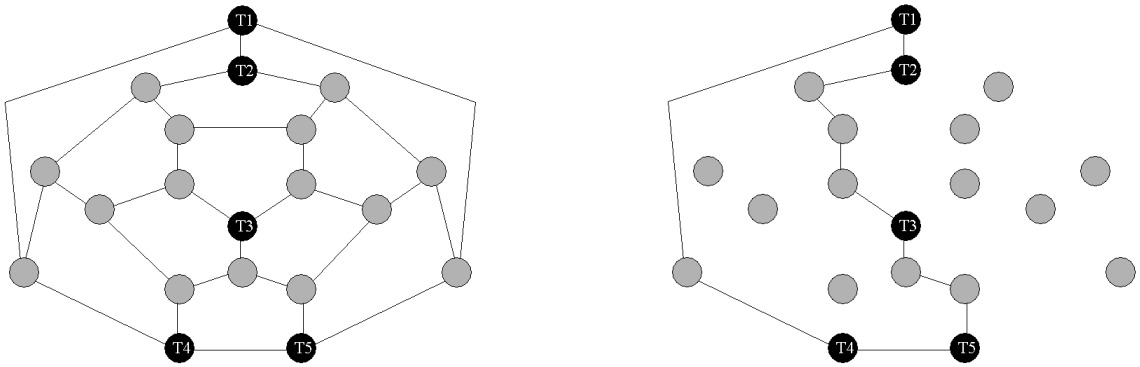
4.2.1.2 Problema 2 (PCf2)



$$R = \{r_{ij} = 4, \forall i, j \in T\}$$

Figura 4.17 Red 4-arista-conexa, basada en una topología de decaedro con dos nodos terminales (izquierda). Los enlaces punto a punto entre nodos de Steiner tienen costo 1, entre un nodo de Steiner y un terminal tiene costo 2, y entre nodos terminales costo 4. Requerimientos de conexión (centro) y solución óptima de costo 20 (derecha). Decaedro extraído de [B1].

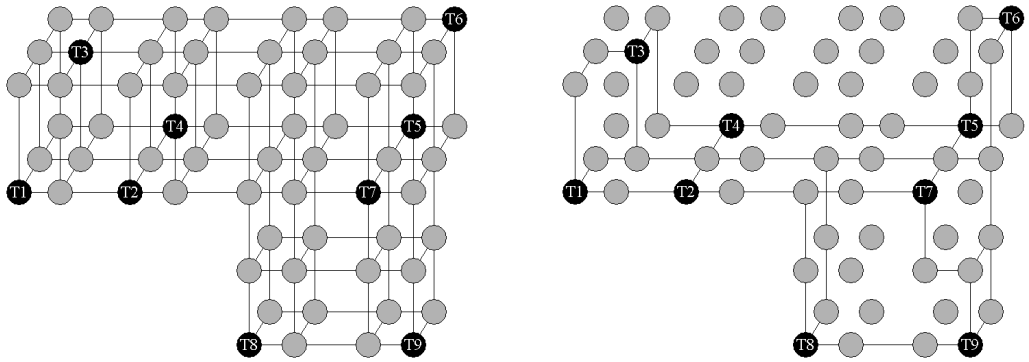
4.2.1.3 Problema 3 (PCf3)



$$R = \{r_{ij} = 2, \forall i, j \in T\}$$

Figura 4.18 Red 2-arista-conexa, basada en una topología de dodecaedro con 5 nodos terminales (izquierda). Los enlaces punto a punto entre nodos de Steiner tienen costo 3, entre un nodo de Steiner y un terminal tiene costo 4, y entre nodos terminales costo 6. Requerimientos de conexión (centro) y solución óptima de costo 45 (derecha). Dodecaedro extraído de [B1].

4.2.1.4 Problema 4 (PCf4)



$$R = \{r_{ij} = 3, \forall i, j \in T\}$$

Figura 4.19 Problema real de diseño de una red de alta confiabilidad, donde se requiere niveles de conexión mayores a 2 entre nodos terminales, es el diseño de una red de transmisión de datos óptica para un portaaviones. Se presenta una topología de red (versión reducida) donde se modela las posibles líneas de comunicación de fibra óptica entre las diferentes estaciones de armamento de un portaaviones (modeladas como nodos terminales). Los costos de los enlaces punto a punto entre nodos de Steiner tienen costo 1, los enlaces entre un nodo de Steiner y un terminal tienen costo 2, y los enlaces entre nodos terminales tienen costo 4 (izquierda). Requerimientos de conexión (centro) y solución óptima de costo 74 (derecha). Problema "Ship" reducido extraído de [B1].

4.2.1.5 Problema 5 (PCf5)

Grafo de gran dimensión con 100 vértices, gran densidad de aristas (400 aprox.) y una amplia gama de requerimientos (10 terminales) de conexión generado aleatoriamente en **[B22]** denominado grafo_2_0, el cual además forma parte de las pruebas de validación (Problema 9) y comparación (Problema 1).

4.2.2 Realización de las pruebas

Para poder determinar los valores más aptos primero es necesario determinar los posibles valores para los parámetros. Los valores utilizados clásicamente son:

- **Cantidad de generaciones:** depende del problema, pero no menos de 100.
- **Tamaño de la población:** cuanto más grande la población mejor ya que tendremos mayor variedad, dependerá esto del problema
- **Tasa de mutación:** varía entre 0.001 y 0.01
- **Tasa de cruzamiento:** varía entre 0.75 y 0.9
- **Tasa de selección:** alrededor de 0.6 (no influye demasiado)
- **Factor de escalado:** varía entre 1.2 y 2 en poblaciones entre 50 y 100 individuos (no influye demasiado)

Algunos de los parámetros, como se puede apreciar, poseen una dependencia mutua muy fuerte con el problema, mientras que los otros no tanto por lo que pueden ser considerados estables. Primeramente podemos observar que el factor de escalado y la tasa de selección no tiene una gran influencia en las soluciones, por lo que con motivo de reducir el número de pruebas a realizar podemos utilizar los valores:

- **Tasa de selección:** 0.6
- **Factor de escalado:** 2 (tendremos poblaciones entre 50 y 150 individuos)

La configuración estará dividida en tres fases:

- Determinación del tipo de cálculo de fitness
- Determinación del tamaño de la población
- Determinación de las tasas de cruza y mutación
- Determinación del criterio de parada (la cantidad de generaciones en caso de utilizarse)

Se procederá de igual manera que en las pruebas de validación, variando uno de los factores de interés y fijando el resto. Los valores por defecto iniciales son:

- **Cantidad de generaciones: 100**
- **Tamaño de la población: 50**
- **Tasa de mutación: 0.003**
- **Tasa de cruzamiento: 0.9**
- **Tasa de selección: 0.6**
- **Factor de escalado: 2**

Si bien los problemas a utilizar son de mediano porte se supone que se pueden extrapolar los resultados obtenidos a cualquier tipo de problema.

Las pruebas de configuración fueron realizadas en una máquina con las mismas características que la de las pruebas de validación.

Los resultados obtenidos pueden apreciarse en el Apéndice 3. y se incluyen en el CD adjunto

4.2.2.1 Cálculo de Fitness

Como primer paso determinaremos que tipo de cálculo de fitness usar. Para realizar esto rápidamente, simplemente correremos el programa para ambas opciones (cálculo de fitness inverso y diferencia) utilizando la representación binaria, para cada uno de los problemas de configuración y observaremos si existe alguna diferencia significativa en las soluciones halladas.

Los resultados obtenidos de las ejecuciones del algoritmo para los problemas de configuración nos muestran que el cálculo de fitness por diferencia de costos produce resultados mejores en cuanto al costo final de la solución.

		PCf1	PCf2	PCf3	PCf4	PCf5
Fitness	**Inverso**	43	25	56	101	1994
	Diferencia	43	20	45	88	1470

Figura 4.20 Costo de las soluciones para las pruebas de configuración del tipo de cálculo de fitness.

Si tenemos en cuenta que el escalamiento lineal no puede ser aplicado en caso de que calculemos el fitness por medio de la inversa del costo (tratamos este tema en el Capítulo 2), podemos descartar este tipo de cálculo para quedarnos con la diferencia.

4.2.2.2 Tamaño de la Población

En vista que este factor depende directamente del tamaño de un individuo tomaremos la representación de mayor dimensión (Binaria) y realizaremos las pruebas solamente para esta representación, buscando así reducir el número de pruebas.

Tomaremos como valores posibles de esta medida 60, 120 y 180. Procederemos realizando una corrida por valor para el problema 5, para el cual se tiene resultados varios. Las medidas de interés a observarse serán la cantidad de memoria utilizada y el tiempo de ejecución. La decisión de qué tamaño deberá tener la población se basa en que es deseable tener la mayor cantidad posible de individuos para mantener la diversidad sin ignorar la memoria utilizada y el tiempo que tarda en realizarse una iteración (lo que depende directamente del tamaño de la población).

Como se puede apreciar en la figura 4.21, las soluciones obtenidas mejoran a medida que el tamaño de la población inicial crece, aunque no de forma sustancial.

	PCf5
60	(509,1468800,500)
120	(498,3048049,500)
180	(492,4729891,500)

Figura 4.21 Resultado de las pruebas de configuración para el tamaño de la población. Se muestra el costo final de la solución, el tiempo de ejecución en milisegundos y la cantidad de iteraciones.

Podríamos optar por utilizar una población con tamaño máximo, aunque como puede verse, el tiempo de ejecución crece proporcionalmente con este (24, 60 y 80 minutos aprox.).

Si observamos en términos de memoria necesaria, las poblaciones poseen un tamaño aproximado de 1.5 Mb, 2.5 Mb y 3.5 Mb (calculado en base a la observación realizada durante la etapa de validación para la representación Binaria) para las poblaciones de 60, 120 y 180 respectivamente. Considerando que durante la ejecución del algoritmo la memoria total no aumenta considerablemente, las tres opciones son manejables.

En base a estas dos observaciones podemos decidir que la utilización de una población inicial de 120 individuos es apta: memoria y tiempo considerables para 500 iteraciones.

4.2.2.3 Tasas de Cruza

Ya que son parámetros sumamente influyentes se toma una mayor cantidad de valores posibles, los cuales son: 0.75, 0.8, 0.85 y 0.9.

Se procede realizando para cada representación un par de pruebas (con cadenas de números aleatorios diferentes) para los problemas de mayor dimensión (Problema 4 y Problema 5) en un número de iteraciones de 300. Como medida de interés se tomará la calidad de la solución.

Ninguno de los valores tomados genera mejores resultados en ninguna de las representaciones, más aún, en muchos casos el tomar un valor u otro no genera diferencia alguna, tal como se puede apreciar en la tabla siguiente.

PCf4	Lista de Caminos	Subgrafo de Requerimientos
0.75	(122,645491, 300,1)	(127,776707, 300,1)
0.8	(122,642742, 300,1)	(127,778754, 300,1)
0.85	(122,645135, 300,1)	(127,785968, 300,1)
0.9	(122,639069, 300,1)	(127,788039, 300,1)

Figura 4.22 Algunos resultados de las representaciones complejas para el problema de configuración 4 de la tasa de cruza. Se muestra el costo de la solución, el tiempo de ejecución, la cantidad de iteraciones y la iteración en la cual se encontró el mejor individuo.

Las diferencias más grandes se pueden apreciar para el problema de mayor dimensión (problema 5), aunque tampoco son tan significativas como para determinar un valor óptimo para la tasa de cruzamiento. Ni siquiera determinan un patrón a seguir para poder observar si al incrementar la tasa, se incrementa o disminuye el costo de la solución o el tiempo de ejecución.

PCf5	Binaria	Lista de Caminos	Subgrafo de Requerimientos
0.75	(955,711201,300,294)	(592,545333,300,278)	(592,676793,300,278)
0.8	(928,725280,300,300)	(574,545169,300,298)	(574,676171,300,298)
0.85	(971,741942,300,297)	(549,545691,300,278)	(549,681286,300,278)
0.9	(914,757366,300,300)	(568,553842,300,278)	(568,685196,300,278)

Figura 4.23 Algunos resultados de las representaciones complejas para el problema de configuración 5 de la tasa de cruza.

Por estas razones, es indiferente en primera instancia la utilización de una u otra tasa por lo que optamos por utilizar una tasa media de 0.85.

4.2.2.4 Tasa de Mutación

Se procede de igual manera que para la tasa de cruza, realizando las mismas pruebas con valores posibles: 0.001, 0.003, 0.006 y 0.01.

La situación correspondiente a la tasa de mutación es diferente a la anterior. La tasa de mutación deja ver sutilmente el patrón de que a medida que esta es mayor, los costos de las soluciones son menores. Esta particularidad se puede apreciar para las estructuras complejas. Se puede deber a que el número de iteraciones utilizado para las pruebas era pequeño, con lo que la mutación "acelera" el proceso. En un proceso normal no es conveniente la utilización de una tasa de mutación grande.

	Lista de Caminos	Subgrafo de Requerimientos
0.001	(652,545826, 300,196)	(525,703071,300,298)
0.003	(568,557989, 300,278)	(525,702549,300,298)
0.006	(558,550601, 300,288)	(506,713734,300,292)
0.01	(542,555287, 300,279)	(506,716629,300,292)

Figura 4.24 Algunos resultados de las pruebas de configuración de la tasa de mutación para el problema 5 utilizando las representaciones complejas.

El caso de la representación binaria es más caótico, no presenta ningún patrón en apariencia, lo que puede ser debido al pequeño número de pruebas realizadas.

	PCf4 (a)	PCf5 (a)	PCf4 (b)	PCf5 (b)
0.001	(84,458276,300, 219)	(1053,772322, 300,296)	(94,484857,300, 106)	(1145,775012, 300,299)
0.003	(88,493305,300, 127)	(956,752602, 300,293)	(102,475531,300, 145)	(1339,777238, 300,271)
0.006	(80,490085,300, 132)	(1109,759725, 300,299)	(89,527542,300, 140)	(930,812726, 300,300)
0.01	(86,517129,300, 171)	(1559,792670, 300,216)	(83,521409,300, 205)	(1446,783615, 300,239)

Figura 4.25 Resultados de las pruebas de configuración de la tasa de mutación para el problema 5 utilizando la representación binaria.

Por tales razones, pensando en un proceso normal con las características definidas hasta el momento y teniendo en cuenta que la diferencia de costos entre las diferentes tasas no es de gran consideración, podemos afirmar que el uso de una tasa de mutación media de 0.003 es apta para nuestro problema.

4.2.2.5 Criterio de parada

Es imposible determinar si usar un criterio de parada por calidad de solución o por cantidad de generaciones para cualquier problema, ya que esto dependerá del problema particular y del tiempo disponible para ejecutar la aplicación. El objetivo será determinar si el criterio por calidad de solución no resulta en una convergencia prematura y si podemos determinar algún número de iteraciones por defecto. Para esto procederemos efectuando una corrida por problema utilizando el criterio de convergencia y uno con una cantidad de generaciones de 1000, para las tres representaciones.

Las pruebas realizadas no muestran diferencias significativas en los resultados para los diferentes criterios de parada, por lo menos para los problemas de pequeña dimensión como el problema 3, tal como lo indica la siguiente tabla:

PCf3		Binaria	Lista de Caminos	Subgrafo de Requerimientos
Criterio	Esfuerzo Pred.	(45,8962,100, 84)	(69,7405,100, 1)	(62,9984,100, 100)
	Calidad	(51,2361,26, 12)	(69,2064,13,1)	(69,2494,13,1)

Figura 4.26 Resultados de las pruebas de configuración del criterio de parada para el problema 3

En estos casos el criterio de esfuerzo predefinido obtuvo mejores resultados. Estos resultados fueron encontrados prácticamente al final de la ejecución, por lo que se puede llegar a pensar en que el número de iteraciones fue insuficiente.

Esto se verifica en los resultados producidos para el problema de mayor dimensión (problema 5), en el cual el número de iteraciones fue mayor (500). En este caso vemos como el criterio de esfuerzo predefinido obtuvo mejores resultados

PCf5		Binaria	Lista de Caminos	Subgrafo de Requerimientos
Criterio	Esfuerzo Pred.	(1470,1300821, 500,147)	(511,801041, 500,25)	(511,1211963, 500,458)
	Calidad	(1669,224687, 84,70)	(551,343332, 93,79)	(551,423986, 93,79)

Figura 4.27 Resultados de las pruebas de configuración del criterio de parada para el problema 5

El número de pruebas no es suficiente para verificar si el criterio de parada por calidad de solución genera una convergencia prematura pero es claro que cuanto mayor sea el número de iteraciones, más probabilidad existe de converger al óptimo.

En base a los tiempos de ejecución del problema 5 y teniendo en cuenta que trabajaremos con poblaciones unas 3 veces más grandes (120 individuos, las poblaciones para estas pruebas fueron de 50 individuos), tomar un criterio de parada por esfuerzo predefinido con 300 iteraciones suena razonable (1/2 hora aproximada de ejecución). Hay que tener en cuenta que para las representaciones complejas la cantidad de iteraciones puede ser menor ya que no poseen la evolución lenta de la representación Binaria.

4.2.2.6 Conclusiones

Si bien durante el análisis de resultados hemos considerado las diferentes representaciones de forma individual, en base a los resultados obtenidos hemos podido tomar decisiones de forma general.

Podemos concluir que luego de realizadas las pruebas, los valores más aptos para la ejecución del algoritmo genético son:

Cálculo de Fitness	Diferencia
Criterio de Parada	Esfuerzo Predefinido
Tamaño de la Población	120
Cantidad de Generaciones	300
Tasa de Cruza	0.85
Tasa de Mutación	0.003
Tasa de Selección	0.6
Factor de Escalado	2

Es bueno notar que en ninguna de las etapas de configuración hemos podido llegar a una conclusión absoluta, en todos los casos hemos decidido un tanto subjetivamente.

4.3 Comparación

Llegados a este punto tenemos realizadas las pruebas de validación y configuración, lo que implica tener el algoritmo genético operativo con los valores más aptos de sus parámetros.

Esta última fase (comparación) intenta analizar en profundidad el desempeño del algoritmo genético realizando una comparación entre las diferentes representaciones del problema, permitiendo concluir finalmente en el "mejor" algoritmo genético adaptado a resolver el Problema Generalizado de Steiner. Además se realizará un análisis comparativo con otras herramientas, tomando como base los resultados obtenidos por estas para los mismos problemas.

4.3.1 Casos de Prueba

Los casos de prueba elegidos pueden dividirse en dos grupos. Primeramente tenemos el problema 8 de validación (PV8) y el problema 4 de configuración (PCf4), ambos de mediana dimensión extraídos de **[B1]**, para los cuales se poseen resultados previos por medio de otra metodología de resolución. En segundo lugar tenemos un conjunto de 3 problemas (denominados en este taller como PCm1, PCm2 y PCm3) de gran dimensión generados aleatoriamente para **[B22]** (denominados allí grafo_2_i con i de 0 a 2), los cuales tienen todos las siguientes características:

- 100 nodos (9 terminales, 91 de Steiner)
- 500 aristas
- 45 requerimientos de conectividad (entre 0 y 8 caminos disjuntos)

Por ser problemas de gran dimensión no es práctico realizar una representación gráfica de estos.

Todos estos grafos se pueden encontrar en el CD adjunto.

4.3.2 Realización de las pruebas

El algoritmo genético provee un conjunto de datos a su finalización, los cuales permitirán la realización de las conclusiones finales. Los datos obtenidos del algoritmo son:

- **Dimensión de la representación:** bytes promedio de un individuo y total de la población.
- **Velocidad de creación de individuos:** tiempo promedio de creación de un individuo y tiempo total de la población en milisegundos.
- **Tiempo de cruza:** tiempo promedio consumido en la realización de una cruza y el total durante la ejecución del algoritmo en milisegundos.
- **Tiempo de mutación:** tiempo promedio consumido en la realización de una mutación y el total durante la ejecución del algoritmo en milisegundos.
- **Tiempo de ejecución:** tiempo total de ejecución del algoritmo y de la evolución en milisegundos (no considera la creación de la población inicial).
- **Cantidad de generaciones:** número de iteraciones realizadas por el algoritmo.
- **Mejor individuo:** Representación del mejor individuo (ver Apéndice 3), iteración en la que se encontró y costo del mismo.
- **Datos generacionales:** en cada iteración se presenta el tiempo transcurrido hasta el momento (milisegundos), el número de generación actual, el mejor fitness de la generación y el fitness promedio de la población.

Procedemos realizando varias corrida por representación para cada problema, variando las cadenas de números aleatorios (10 secuencias desde la A a la J), desplegando los datos obtenidos y graficando la información generacional en dos gráficas que representan la evolución del fitness, tanto el mejor como el promedio, en el tiempo y en el número de generaciones.

Las pruebas fueron realizadas en una máquina con las mismas características que la de las etapas anteriores.

Los resultados obtenidos pueden apreciarse en el Apéndice 3. y se incluyen en el CD adjunto

4.3.2.1 Comparación entre Estructuras

La comparación entre estructuras puede ser llevada en varios términos:

- **Estructura:** comparando tamaño de los individuos y complejidad de los operadores.
- **Extensión:** simplicidad en el manejo de estas y posible extensión de los operadores para lograr un mejor desempeño.
- **Tiempo:** tiempos de creación de individuos, ejecución de los operadores genéticos, y de ejecución del algoritmo.
- **Población Inicial:** características de la población inicial.
- **Desempeño:** convergencia, mejores y peores soluciones, robustez, evolución, etc.

Con excepción del último punto, todos pueden ser tratados con un enfoque general (para cualquier problema). El último depende del tipo de problema en el cual se realizan las observaciones, tal como se verá más adelante.

Estructura

No podemos hablar de las representaciones complejas por separado, ya que estas presentan exactamente los mismos resultados, debido a que los operadores son exactamente iguales.

Como se hizo notar en la etapa de validación, la representación binaria es la más grande en término de memoria consumida por individuo debido a la representación de todas las aristas del grafo.

La complejidad de los operadores es similar, la representación binaria presenta la comlpejidad extra del chequeo de factibilidad en sus operadores, mientras que las representaciones complejas necesitan realizar un esfuerzo extra en la creación de individuos.

Extensión

Durante la etapa de implementación, la estructura que mayor dificultad planteó fue la de Lista de Caminos con la heurística de selección de caminos. Sin embargo, esta estructura, así como la de Subgrafo de Requerimientos son las que más posibilidades ofrecen a la hora de plantear diferente implementación de los operadores genéticos entre los cuales podemos indicar:

- Creación de individuos con diferentes heurísticas
- Cruzamiento eliminando y/o agregando aristas de todas las restricciones no en una restricción particular.
- Mutar un gen hallando nuevos caminos para una restricción, teniendo en cuenta las aristas ya existentes.

Tiempo

En cuanto a tiempos se refiere, las representaciones complejas presentan un menor tiempos en la realización de la operación de cruza como en el tiempo global de ejecución del algoritmo, no siendo significativa esta diferencia en el último caso.

	Binaria	Lista de Caminos	Subgrafo de Requerimientos
Creación de Individuos (ms)	32.1	59.5	76.8
Cruza (ms)	2444.9	28.5	371
Mutación (ms)	4240.7	5270.2	4395.2
Ejecución (min)	34	26	24

Figura 4.28 Tiempo promedio de creación, cruza, mutación y ejecución del algoritmo para las diferentes representaciones durante las pruebas de comparación de los problemas PCm1, PCm2 y PCm3.

Podemos ver que la representación Binaria, sin embargo, presenta un mejor desempeño en el tiempo de creación de individuos.

No existen diferencias sustanciales en cuanto al operador de mutación ya que mientras la representación binaria necesita chequear la factibilidad, las representaciones complejas necesitan hallar nuevos caminos a partir de un grafo aleatorio.

Población Inicial

En cuanto a la calidad de los individuos vemos como las representaciones complejas generan individuos con un fitness considerablemente superior, debido esto al algoritmo de selección de caminos, lo cual realiza la creación de forma más “inteligente” que la creación aleatoria en la que se basa la representación Binaria.

	Binaria	Lista de Caminos	Subgrafo de Requerimientos
Fitness	1018.9250	4133.0498	4149.2168

Figura 4.29 Muestreo de calidad en individuos promedio, integrantes de la población inicial para el problema PCm1.

Desempeño

Comencemos dejando en claro la robustez del algoritmo en cualquier caso, en donde por robustez consideramos la correcta evolución de éste, generando resultados medianamente aceptables.

En este punto debemos considerar dos situaciones:

- problemas aleatorios de gran dimensión (PCm1, PCm2 y PCm3)
- problemas estructurados de mediana-gran dimensión (PV8, PCf4) provenientes de situaciones reales

Estas dos situaciones mostraron diferentes comportamientos del algoritmo según la estructura, por lo que es conveniente analizarlas por separado.

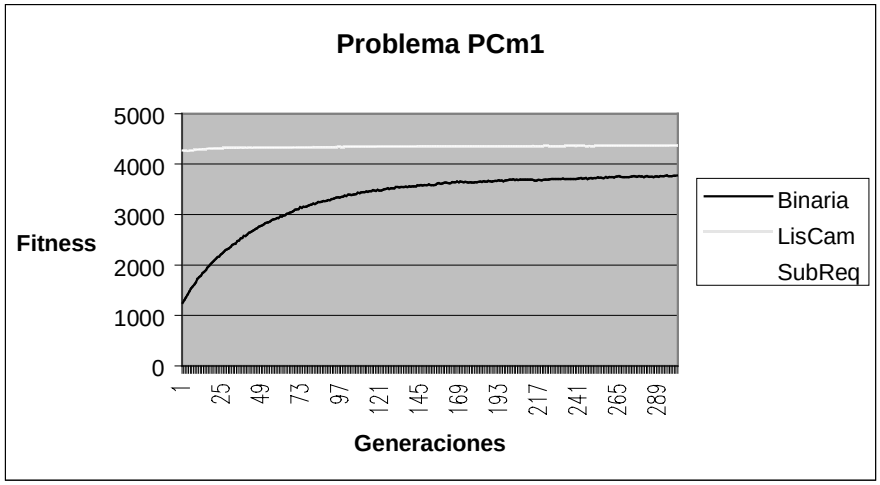
Problemas Aleatorios

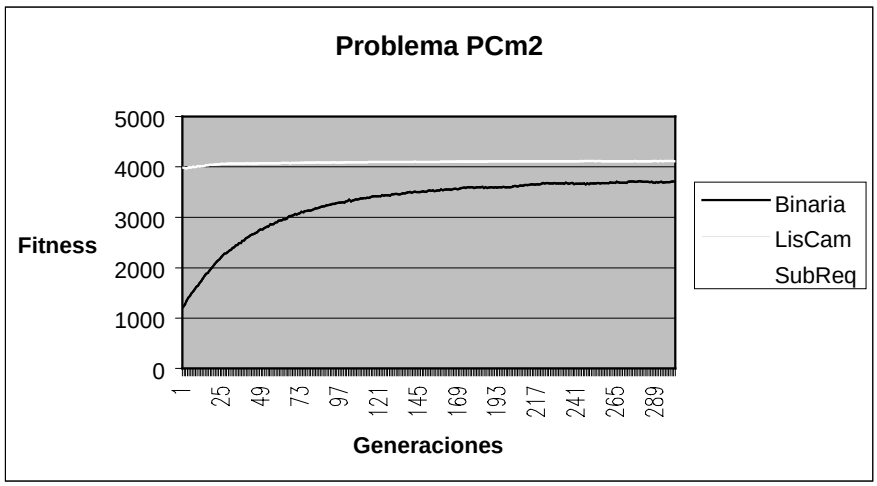
Proyectando las ejecuciones promedio para cada uno de los problemas PCm1, PCm2 y PCm3, podemos analizar más profundamente las diferencias y similitudes entre las estructuras.

Como se puede apreciar en la figura 4.30, si bien la convergencia de ambas representaciones es similar, las representaciones complejas llegan a valores mucho más rápido, muchas veces en la primer generación. Esto produce una evolución lenta del algoritmo.

Los mejores valores han sido hallados con las representaciones complejas y como se ha observado antes la representación binaria está lejos de estos valores en gran parte de los casos.

La comparación es un tanto injusta ya que la cantidad de iteraciones para la representación binaria pudo haber sido insuficiente, debido a su lenta convergencia. Además debemos considerar que las representaciones complejas generan muy buenos individuos para la población inicial, por lo que los buenos resultados no pueden ser adjudicados a la evolución genética en sí, como en el caso de la representación Binaria.





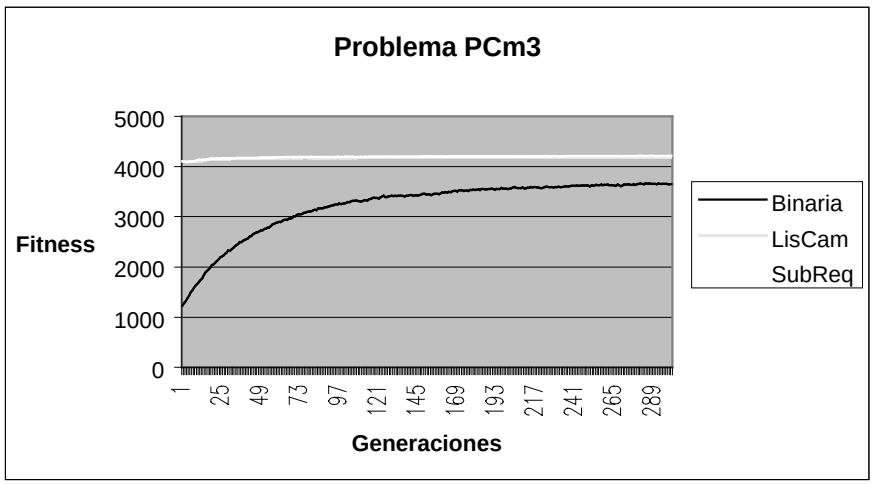


Figura 4.30 Promedio de ejecuciones para las tres representaciones sobre los problemas PCm1, PCm2 y PCm3.

Problemas Estructurados

La situación en los problemas estructurados (PV8 y PCf4) es totalmente diferente. Tal como podemos apreciar en la figura 4.31, la representación binaria es la que halla los mejores resultados, además de estar muy cerca del mejor valor conocido.

Las representaciones complejas presentan un comportamiento extraño, generan buenas soluciones pero no tan cercanas a la mejor solución conocida y hallan en general su mejor solución en las primeras generaciones (las primeras 5). A partir de ahí el algoritmo entra en una etapa de estancamiento en la cual no se detecta evolución, por mínima que sea.

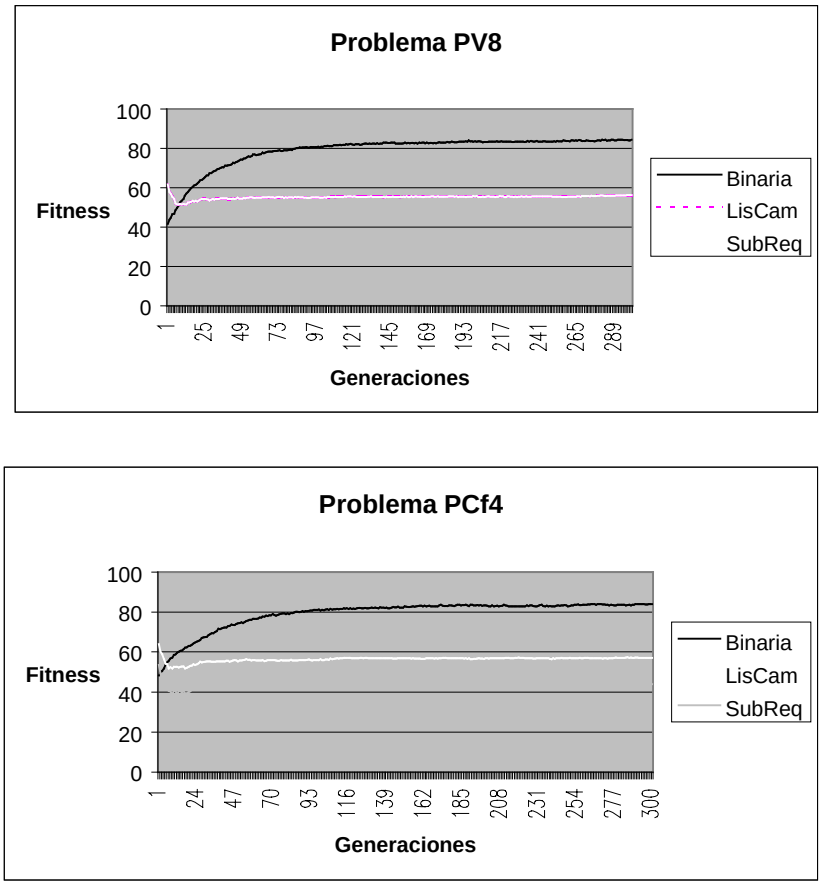


Figura 4.31 Promedio de ejecucion para las tres representaciones sobre los problemas PV8 y PCf4

4.3.2.2 Comparación con Resultados Anteriores

Las referencias que se tienen de la utilización de algoritmos genéticos a la resolución del Problema Generalizado de Steiner son pocas, tal como se ha hecho explícito antes, más aún resultados concretos. Los resultados que aquí trataremos provienen de **[B1]**, trabajo en el cual se utilizó la heurística denominada Ant System sobre los grafos de prueba PV8 y PCf4, (problemas 8 y 4 de validación y configuración respectivamente) y también de **[B22]**, proyecto en el cual se trató el problema con algoritmos genéticos distribuidos y se aplicó la herramienta construida a los problemas PCm1, PCm2 y PCm3 (problemas 1, 2 y 3 de la etapa de comparación correspondientes a los grafos: grafo_2_0, grafo_2_1 y grafo_2_3). Se tiene además el menor costo hallado hasta el momento para el grafo_2_0 por integrantes de esta línea de investigación, el cual es de 358.

Ant System

Comencemos viendo los resultados que obtuvo la ejecución del algoritmo basado en la meta heurística Ant System para los problemas PV8 y PCf 4.

Problema	Porcentaje de desviación de la solución óptima	Resultado obtenido (aproximado)	Costo de la mejor solución conocida	Tiempo de ejecución
PCf4	**0**	**74**	**74**	**38.20**
PV8	**3.92**	**102**	**98**	**43.32**

Figura 4.32 Resultados obtenidos por la meta heurística Ant System para los problemas PCf4 y PV8 de [B1].

Estos resultados nos muestran soluciones cercanas al óptimo u el óptimo mismo con un tiempo de cálculo de aproximadamente 40 minutos ambos. Cabe aclarar que estos resultados fueron los mejores hallados, en otros casos se llegaron a soluciones hasta un 10 % peores.

Esta heurística es bastante inestable ya que al contrario del algoritmo genético que hemos desarrollado y tal como lo hemos observado, depende muy fuertemente de los valores de sus parámetros de configuración, en definitiva no es un algoritmo tan robusto y muy proclive a resultados lejos del óptimo.

Sin embargo podemos ver que los resultados del algoritmo genético nunca fueron óptimos y en varios casos los resultados estuvieron lejos de este para cualquiera de los problemas. Estos casos corresponden a la corrida de las representaciones complejas.

Para el problema PCf4, podemos ver que la representación binaria posee la media de resultados más próxima de 84.3 con un costo mínimo de 83 hallado en 20 minutos aproximadamente. La situación de las representaciones complejas es totalmente diferente, estando estas muy lejos de la solución con una solución cuyo costo promedio es de 115.2 no teniendo ninguna de estas un costo inferior a 110.

		Binaria	Lista de Caminos	Subgrafo de Requerimientos
Mejor Solución	Costo	83	110	110
	Tiempo	~20 min	~27.5 min	~33.1 min
Peor Solución	Costo	88	126	125
	Tiempo	~21.3 min	~21.6 min	~25.8 min

Figura 4.33 Mejores y peores soluciones para el problema PCf4.

Para el problema PV8, la situación cambia, ya que lo que se presentó como solución óptima en la figura 4.33, no lo era tal y la representación binaria obtuvo con un costo promedio de 97.9 un mejor resultado de 93 en 37 minutos aproximadamente- Para las representaciones complejas la situación continuó incambiada, se obtuvo un costo promedio de 120.3 con costos no menores a 116.

		Binaria	Lista de Caminos	Subgrafo de Requerimientos
Mejor Solución	Costo	93	116	116
	Tiempo	~37.2 min	~63.4 min	~81 min
Peor Solución	Costo	103	126	126
	Tiempo	~40.8 min	~48.5 min	~58.5 min

Figura 4.34 Mejores y peores soluciones para el problema PV8.

En general podemos observar que se producen resultados parecidos para ambas metodologías, si utilizando la representación binaria en el algoritmo genético.

Algoritmos Genéticos Paralelos

La resolución del Problema Generalizado de Steiner mediante un enfoque distribuido **[B22]**, se realizó en base a la comparación de diferentes modelos de paralelización.

Los resultados primarios mostraban que el modelo paralelo no resultaba mejor sin considerar ciertas características de paralelización. Estos resultados mostraban en promedio para los grafos PCm1, PCm2 y PCm3, que tanto el algoritmo serial como el paralelo hallaban soluciones con un costo del entorno de los 730 (fitness diferencia de ~4200). Este resultado fué calculado en base al promedio de fitness por iteración para cada problema.

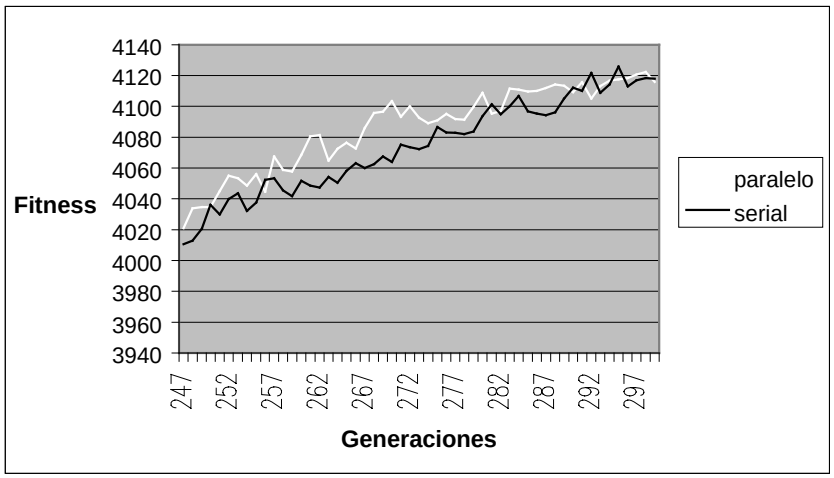


Figura 4.35 Resultados promedio de las corridas para los cinco primeros problemas y comparación entre los modelos serial y paralelo de [B22].

Un hecho interesante a tener en cuenta es que los valores de los parámetros de la ejecución del algoritmo genético en **[B22]** son los mismos que los de este proyecto (120 individuos, 300 generaciones, tasa de mutación de 0.003 y factor de escalado de 2).

Promediando los resultados obtenidos durante nuestra experimentación (para los tres primeros problemas) podemos ver que nuestro algoritmo genético tiene un desempeño superior al anterior en el caso de las representaciones complejas, con un costo promedio de solución del orden de los 1069.9 para la representación Binaria, 602.8 para la de Lista de Caminos y 599.5 para la de Subgrafo de Requerimientos. El comportamiento promedio del algoritmo pueden observarse en la figura 4.30.

Resultados posteriores, ya con la aplicación de diferentes modelos de paralelización muestran el mejor desempeño del algoritmo paralelo en términos de tiempo. En particular el modelo denominado "Migración" no presenta una superioridad significativa en términos de costo final pero sí en cuanto al tiempo de ejecución para llegar a esa solución. Esto se puede ver claramente en la siguiente figura.

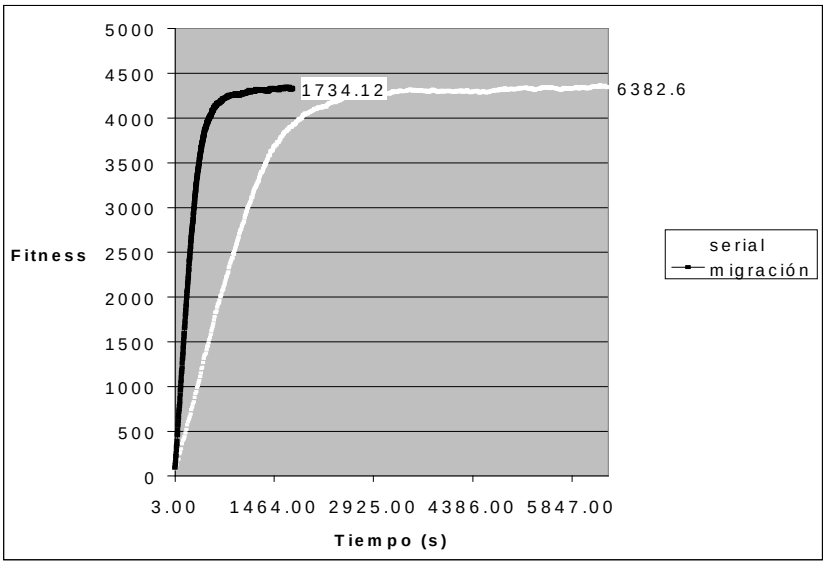


Figura 4.36 Resultados promedio de las corridas para el modelo de "Migración" y el serial en [B22].

Por nuestro lado, vemos que obtenemos resultados parecidos en las representaciones complejas. La representación binaria no presenta tan buenos resultados como para hacerla comparable con estos.

Otros Resultados

Comparando el mejor resultado hallado hasta el momento para el problema PCm1 (costo de solución de 358) podemos ver que las mejores soluciones halladas durante el transcurso del proyecto, no están lejos del mismo.

		Binaria	Lista de Caminos	Subgrafo de Requerimientos
Mejor Solución	Costo	745	470	470
	Tiempo	162 (iter. 1555 de 2000)	34 (iter. 939 de 1000)	41 (iter. 939 de 1000)

Figura 4.37 Mejores resultados hallados en el transcurso del proyecto para el problema PCm1. Los tiempos están en minutos.

En el caso de la representación binaria se nota una mejoría significativa al incrementar el número de generaciones, no tanto así para las representaciones complejas (ver Apéndice 3, resultados con 2000 iteraciones).

Capítulo 5 Conclusiones

Finalizada la etapa de experimentación, varias conclusiones han aparecido, así como problemas que escapando del alcance del proyecto son fuertes candidatos para resolverse en trabajos futuros. Aquí presentaremos de forma más ordenada las conclusiones que se desprenden del proyecto así como las pautas necesarias para la realización de esos trabajos futuros.

5.1 Conclusiones

Metodología

La primer conclusión que podríamos sacar es acerca de la metodología a seguir en caso de intentar resolver el Problema Generalizado de Steiner con Algoritmos Genéticos. La gran variedad de problemas y parámetros a configurar del algoritmo hacen imposible tomar decisiones antes de enfrentarse al problema y realizar un análisis preliminar del mismo.

Por análisis preliminar nos referimos a considerar la cantidad de requerimientos que este tiene, la densidad de aristas, etc. Esto nos permitirá definir entre otras cosas algunos parámetros tales como la probabilidad de selección de una arista para crear los individuos aleatoriamente. Por ejemplo, si tenemos un margen considerable entre la cantidad de aristas que pueden formar una solución y la cantidad de aristas del grafo, podemos tomar una probabilidad relativamente baja para así comenzar con una población inicial de individuos con gran fitness.

Un segundo y último paso antes de la aplicación de la herramienta, sería la realización de pruebas de configuración particulares al problema. Si bien se ha podido observar durante la etapa de configuración que algunos parámetros tales como la tasa de cruzamiento, factor de escaladao y tasa de selección no afectan considerablemente las soluciones si tomamos casos genéricos, habría que observar la influencia del resto de los parámetros. Por ejemplo: si en determinados casos se llega rápidamente a un estancamiento, este podría llegar a ser evitado usando una tasa de mutación más grande, o quizás variable.

Este análisis previo provee de un mejor contexto de trabajo y acota en cierta medida lo aleatoria que pueda resultar la resolución del problema.

Representaciones

Existió tanta variedad de resultados que no es posible afirmar qué tan prometedora pueda ser una representación u otra, ni descartar el uso de alguna, teniendo en cuenta principalmente que:

- el conjunto de pruebas que se ha elegido así como las secuencias de números aleatorios pudieron no haber sido adecuadas ni suficientes
- se mostraron dos contextos (problemas estructurados y aleatorio) que muestran resultados contrarios
- resultados empíricos realizados en otros trabajos **[B22]**, así como la bibliografía básica **[B1]** expresan que la representación Binaria es la qué más aplica a cualquier tipo de problema a resolverse con esta metodología
- se generaron resultados similares en las estructuras complejas y no se profundizó en la realización de operadores particulares para cada una de ellas que permitieran una comparación más exhaustiva

En base a las pruebas realizadas, podemos afirmar que:

- las representaciones se comportan de forma diferente según el contexto de aplicación (problemas estructurados y aleatorios)
- las representaciones complejas se comportan mejor en problemas aleatorios
- las representación Binaria se comporta mejor en problemas estructurados
- las representaciones complejas presentan en general buenos individuos en la etapa de inicialización del algoritmo y una lenta evolución
- la representación binaria presenta un comportamiento similar a su aplicación en otros problemas de optimización, esto es: una rápida evolución en las primeras iteraciones del algoritmo y una lenta pero continua evolución de ahí en más
- el tiempo de ejecución es similar para todas las representaciones
- es necesario experimentar con otros operadores genéticos así como corrección de individuos

Otros Resultados

Si bien no podemos hablar ampliamente de una comparación con otras metodologías, por no tener información suficiente acerca de otras técnicas aplicadas a este problema más que las conocidas, podemos afirmar que comparado con los resultados de los trabajos conocidos [B1,B22], el algoritmo genético se desempeño satisfactoriamente.

5.2 Trabajo Futuro

Dentro de las líneas de trabajo futuro debemos considerar los siguientes puntos:

- **Motor Genético:** Cómo se ha mencionado, el motor utilizado es un fragmento adaptado del motor que ha estado siendo desarrollado por el CeCal con algunas diferencias y agregados que engrosan las funcionalidades del primero. Una línea a seguir es la unión de estos motores con la obtención de uno con las siguientes características:

    - **Motor configurable:** se debe permitir la configuración de parámetros así como de las diferentes opciones de selección, cruza, mutación, cálculo de fitness y criterio de parada sin necesidad de re compilar el motor.
    - **Variedad de operadores:** operadores genéticos ampliados, los cuales deben incluir selección por torneo y ruleta, cruza de 1 y 2 puntos, cruza multipunto, mutación simple, etc.
    - **Chequeo de Factibilidad:** se debe permitir el chequeo de la factibilidad de un individuo luego de la acción de un operador genético, en caso de que sea necesario.
    - **Ambiente:** incluir la idea de "ambiente de ejecución" el cual no es más que la conjunción de los datos globales, parámetros del algoritmo, generador de números aleatorios y cualquier otra cosa que necesite ser accedida por cualquier operación.
    - **Estadísticas:** recolectar datos variados de la ejecución del algoritmo en una estructura para permitir luego de finalizado este la realización de estadísticas y observaciones necesarias.

- **Cruzamiento de dos puntos:** se debería experimentar con la realización de la cruza de dos puntos. En el caso de la representación binaria, si el segmento que se quita al genotipo es proporcionalmente pequeño resulta en una mutación potenciada (mutar varios genes contiguos). El resultado es variar varias aristas de uno o dos vértices solamente, en lugar de estar retirando aristas aleatoriamente de cualquier lado.

- **Corrección de individuos:** hasta el momento se ha estado descartando los individuos no factibles, lo que produce un descenso en el desempeño del algoritmo cuando esto empieza a ser algo común (cuando la población tiende a estabilizarse). Sería conveniente considerar alguna técnica de corrección que permita factibilizar esos individuos de manera rápida y así reducir tiempos.

- **Operadores complejos:** los operadores utilizados en las representaciones complejas han sido los mismos, lo que produjo que los resultados sean similares, teniendo en cuenta que la creación de individuos es la misma. Se pudo observar además que la replicación de información de aristas provoca que los operadores de cruza y mutación no representen cambios sustanciales en los individuos generados. Otros operadores pueden ser desarrollados, como por ejemplo: realizar la mutación transformando los individuos a su representación Binaria, realizar la mutación Binaria y volverlos a su representación original, de forma tal de agregar o eliminar una arista de toda la representación y no de una restricción particular.

- **Mutación dinámica:** un problema detectado fue el estancamiento de la representación binaria antes de llegar a valores considerables en gran parte de las pruebas. Un punto a tratar sería la utilización de una tasa de mutación dinámica, de forma tal de aplicar más seguido la mutación en caso de producirse este estancamiento e intentar salvar esa brecha.

- **Comparación formal:** las comparaciones realizadas han sido en base a la observación de los resultados en bruto que las diferentes pruebas han proporcionado. Para tener una mejor medida de qué tan parecidas son las representaciones, se podría realizar un análisis estadístico más profundo.

- **Análisis Formal:** se podría orientar una línea de investigación qué haga énfasis en el GSP y la detección de patrones o metodologías orientadas a la posterior aplicación de AG en un entrono más apto. Hasta el momento los trabajos con AG aplicados a este problema estuvieron enfocados en la herramienta y la observación de resultados empíricos de su aplicación.

- **Otras metodologías:** es deseable tener resultados de la aplicación de este problema con otras heurísticas, más allá de las vistas hasta el momento para realizar una comparación más profunda no solo de la herramienta sino también de la técnica en general.

Referencias

Bibliografía

Problemas de Steiner

[B1] F. Robledo, "Diseño Topológico de Redes", Tesis de Maestría (2000), Instituto de Computación Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay.

[B2] B.Awerbuch., Y.Azzar., Y.Bartal, "On-Line Generalized Steiner Problem", Proc. of 7th SODA, pp.68-74, (1996), URL: www.math.tau.ac.il/~azar/gst.ps (Febrero 2003)

[B3] U.Fößmeier., M.Kaufmann, A.Zelikovsky, "Faster Approximation Algorithms for the Rectilinear Steiner Tree Problem", Proc. 4th Int. Symp. Algorithms & Computation, Lecture Notes in Computer Science, Volumen 762, pp. 533-542, (1993)

[B4] Y.Kusakari, D.Masubuchi, T.Nishizeki, "Algorithms for Finding Noncrossing Steiner Forests in Plane Graphs", Lecture Notes in Computer Science, Volumen 1741, pp 03-37, (1999)

[B5] C.Gröpl, S.Hougardy, T.Nierhoff, H.Prömel, "Lower Bounds for Approximation Algorithms for the Steiner Tree Problem", Lecture Notes in Computer Science, Volumen 2204, pp 0217, (2001)

[B6] K.Jain, "A Factor 2 approximation algorithm for the generalized Steiner network problem" Combinatorica, ISSN: 0209-9683, Volumen 21 Issue 1 (2001) p.39-60

[B7] A.Agrawal, P.Klein, R.Ravi, "When trees collide: An approximation algorithm for the generalized Steiner problem in Networks", Proceedings of the 23rd ACM Symposium on Theory of Computing, pp.134-144 (1991)

[B8] D.P.Williamson, M.X.Goemans, M.Mihail, V.V.Vazirani, "A Primal-Dual approximation algorithm for Generalized Steiner Network problems" Combinatorica, Volumen 15, Número 3, p.435-454 (1995)

[B9] S.VoB, "Problems with Generalized Steiner problems", Algorithmica, Volumen 7, Números 2&3, p.333-335 (1992)

[B10] P.Winter, "Generalized Steiner problems in outerplanar graphs", Networks, p.129-167 (1987)

[B11] M.Stoer, "Design of Survivable Networks", Lecture Notes in Mathematics, ISBN 3-540-56271-0, ISBN 0-387-56271-0, Springer-Verlag (1996)

Algoritmos Genéticos y Técnicas evolutivas

[B12] G. E. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning". Addison Wesley, New York, (1989).

[B13] K.A.De Jong, W.M.Spears, "Using genetic algorithms to solve NP-Complete problems", Proc. of the Third Int. Conf. on Genetic Algorithms, 124-132, (1989) URL: citeseer.nj.nec.com/dejong89using.html (Febrero 2003)

[B14] M.Guervos, "Informatica Evolutiva", URL: kal-el.ugr.es/~jmerelo (Febrero 2003)

[B15] D.Beasley, D.R.Bull, R.R.Martin, "An overview of Genetic Algorithms Part1, Fundamentals" University Computing, (1993), 15(2), pp.58-69, URL: www.geocities.com/francorbusetti/gabeasley1.pdf (Febrero 2003)

[B16] D.Beasley, D.R.Bull, R.R.Martin, "An overview of Genetic Algorithms Part2, Research Topics" University Computing, (1993), 15(4), pp.170-181 URL: www.geocities.com/francorbusetti/gabeasley2.pdf (Febrero 2003)

[B17] D.Whitley, "A Genetic Algorithm Tutorial", Computer Science Department, Colorado State University, Statistics and Computing, vol. 4, pp.65-85, (1994) URL: citeseer.nj.nec.com/whitley93genetic.html (Febrero 2003)

[B18] C.A.Coello, "La Importancia de la Representación en los Algoritmos Genéticos" Soluciones Avanzadas, Año 7, No 69, pp.50-56, (1999), (Parte 1) Soluciones Avanzadas, Año 7, No 70, 1999, pp.44-48, (Parte 2) URL: delta.cs.cinvestav.mx/~ccoello/genetic.html (Febrero 2003)

[B19] M.Nowostawsky, R.Poli, "Parallel Genetic Algorithm Taxonomy" URL: citeseer.nj.nec.com/nowostawski99parallel.html (Febrero 2003)

[B20] S. Nesmachnow, "Evolución en el diseño de Algoritmos Genéticos Paralelos" Instituto de Computación (InCo). Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay, CLEI 2002 - Reporte INCO 06-02, Julio 2002 URL: http://www.fing.edu.uy/~sergion/gp/documentos.html (Febrero 2003)

[B21] S. Nesmachnow, "Diseño de Redes de Comunicaciones Confiables" Instituto de Computación (InCo). Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay, Trabajo de investigación interna, Agosto 2002 URL: http://www.fing.edu.uy/~sergion/gp/documentos.html (Febrero 2003)

Aplicación de AG a los Problemas de Steiner

[B22] S.Arraga, M.Aroztegui, "Algoritmos Genéticos Paralelos para el Problema General de Steiner en Grafos", Proyecto de Grado (2002), Instituto de Computación (InCo). Facultad de Ingeniería Universidad de la República, Montevideo, Uruguay.

[B23] S.Voss, K.Gutenschwager, "A Chunking Based Genetic Algorithm for the Steiner Tree Problem in Graphs", DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volumen 40, 1998, 335-3555

[B24] H.Esbensen, "Computing Near-Optimal Solutions to Steiner Problem in a Graph Using a Genetic Algorithm", Proc. of The European Design and Test Conference, Paris, France. Febrero 1994, URL: www.daimi.au.dk/PB/468/PB-468.pdf (Febrero 2003)

[B25] R.Huang, J.Ma, D.F.Hsu, "A Genetic Algorithm for Optimal 3-connected Telecommunication Network Designs", IEEE - Proceedings of the 1997 International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN `97)

Varios

[B26] L.R.Ford, Jr. and D.R.Fulkerson, "Flows in Networks", Princeton University Press, Princeton, NJ, (1962)

[B27] M.A.Weiss, "Estructuras de Datos y Algoritmos", Addison-Wesley Iberoamericana, ISBN 0-201-62571-7, (1995)

[B28] B.W.Kernighan, D.M.Ritchie, "The C Programming Language", Prentice-Hall, Inc., ISBN 0-13-110163-3, (1978)

[B29] C.Larman, "UML y Patrones", Prentice Hall, Inc., ISBN 970-17-0261-1 (1999)

[B30] D.E. Knuth, "The Art of Computer Programming, Volume 2, Seminumerical Algorithms, Third Edition, Addison-Wesley, ISBN 0-201-89684-2 (1997)

Enlaces

Algoritmos Genéticos

[E1] Algoritmos Genéticos Paralelos y su Aplicación al Diseño de Redes de Comunicaciones Confiables, S. Nesmachnow, Proyecto CSIC
URL: http://www.fing.edu.uy/~sergion/gp.html (Febrero 2003)

[E2] Genetic Algorithms Warehouse
Repositorio de información sobre Algoritmos Genéticos.
URL: http://geneticalgorithms.ai-depot.com (Febrero 2003)

[E3] The Genetic Algorithms Archive
Repositorio de información relacionado con la investigación en el área de los Algoritmos Genéticos
URL: http://www.aic.nrl.navy.mil/galist/ (Febrero 2003)

[E4] Illigal (Illinois Genetic Algorithms Laboratory)
Grupo de investigación en algoritmos evolutivos.
URL: http://www-illigal.ge.uiuc.edu/index.php3 (Febrero 2003)

Buscadores

[E5] The Guide for Electronic Theses & Dissertations
Guía electrónica de tesis y disertaciones, muchas de ellas relacionadas con Ciencias de la Computación
URL: http://etdguide.org/ (Febrero 2003)

[E6] CiteSeer (Nec Research Institute)
Biblioteca digital de literatura científica
URL: http://citeseer.nj.nec.com/cs (Febrero 2003)

[E7] NDLTD (Networked Digital Library of Theses and Dissertations)
Biblioteca digital de literatura científica
URL: http://www.ndltd.org/ (Febrero 2003)

[E8] The Computing Research Repository (CoRR)
Repositorio de información sobre Ciencias de la Computación
URL: http://xxx.lanl.gov/archive/cs/intro.html (Febrero 2003)

[E9] DBLP Bibliography
Biblioteca digital de literatura sobre Ciencias de la Computación
URL: http://www.informatik.uni-trier.de/~ley/db/ (Febrero 2003)

Varios

[E10] Programming in C
A.D.Marshall, Programación en C, llamadas al sistema UNIX y subrutinas usando C
URL: http://www.cs.cf.ac.uk/Dave/C/CE.html (Febrero 2003)

Apéndice 1 Estudio del Problema de Steiner

El diseño de una red tanto de telecomunicaciones como carretera o energética es analizado usando varias medidas como costos de construcción, tiempo de retardo, confiabilidad y políticas de ruteo. Dadas las ubicaciones de los nodos y otra información como costos, distancias y trafico entre los nodos, el problema principal estará en determinar la topología que minimice el costo total de conexión bajo ciertas restricciones estipuladas.

Una de las restricciones fundamentales de diseño, es la confiabilidad de la red, básicamente, permitir que la misma siga siendo operativa ante una caída de una línea de comunicación o de la falta de algún nodo, cualquiera sea este. Esto se logra agregando redundancia en la cantidad de caminos disjuntos que puedan existir entre todo par de nodos. Este problema es conocido como el problema generalizado de Steiner.

Una de las técnicas que mejores resultados ha obtenido ha sido la de Algoritmos Genéticos, la cual no incluimos en este informe.

En la siguiente sección definiremos el problema de Steiner y sus variantes más conocidas y por último nos concentraremos en la resolución del problema generalizado.

A1.1 Definición del Problema

Podríamos modelar la red como un grafo y pensar que el problema general es minimizar el costo total de las aristas, manteniendo conexos un subconjunto de nodos del grafo original y satisfaciendo ciertos requerimientos de conectividad entre esos nodos. Este problema ha sido denominado Problema de Steiner.

Existen variados casos dependiendo básicamente de estos requerimientos de conectividad y de otros requerimientos sea tanto para simplificar el modelo como para lograr una mayor generalización (costos asociados a aristas y/o vértices, dirección o indirección de las aristas, grafos planares o no, etc). Cada nueva restricción agregada lleva a definir una nueva subclase del problema de Steiner.

A continuación se describirán algunos de los más importantes, los que a su vez han sido mayormente tratados.

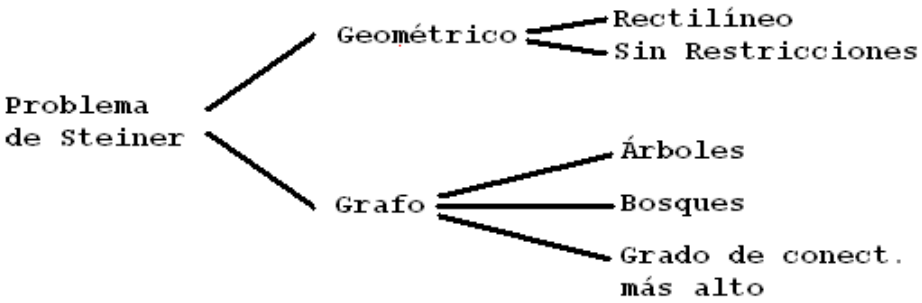


Figura A1.1 Clasificación del problema de Steiner.

A1.1.1 Problema de Steiner en Grafos

Sea $G = (V,E)$ un grafo no dirigido conexo, con $n = |V|$, c una matriz de costos asociada a las aristas, y $T \subseteq V$, con $n_T = |T|$, un conjunto de nodos a los cuales se les denomina terminales. Se define el Problema de Steiner en Grafos (o Redes, Steiner Problem in Networks - SPN) que consiste en ncontrar un subgrafo G_T de G tal que entre todo par de nodos de T existe un camino en G_T que los une, y además el costo del subgrafo G_T es mínimo. A los nodos de (V\T) se le denomina nodos de Steiner.

Las aplicaciones de este problema son variadas, diversos problemas de diseño de redes de comunicaciones pueden ser modelados como un SPN.
Casos especiales del SPN y reducciones se pueden encontrar en **[B1]**, así como algoritmos exactos y aproximados para su resolución.

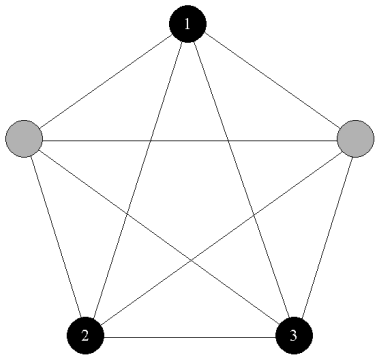


Figura A1.2 Grafo a utilizar en los ejemplos. Los nodos negros numerados son los terminales, y los grises, los de Steiner. El costo de las aristas está dado por el largo de estas.

Problema del Árbol de Steiner (Steiner Tree Problem - STP)

El árbol de Steiner de un conjunto de vértices del grafo G es un subgrafo conexo de mínimo peso de G que incluye todos los vértices. Es siempre un árbol. Los árboles de Steiner tienen aplicaciones practicas, como por ejemplo, en la determinación de largo total mínimo de cable necesario para conectar un conjunto de puntos.

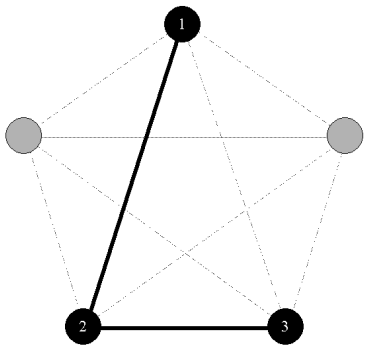


Figura A1.3 Solución óptima del problema del árbol de Steiner para el grafo de ejemplo.

Problema del arbol dirigido de Steiner (Directed Steiner tree problem)

Es el problema del árbol pero con la restricción de que las aristas del grafo son dirigidas.

Problema del bosque de Steiner (Steiner Forest Problem)

Bajo las mismas condiciones planteadas en la definición del SPN y k subconjuntos disjuntos, un bosque de Steiner en G es un conjunto de k árboles en G tal que cada árbol conecta todos los vértices de un subconjunto diferente. Por ser conjuntos disjuntos, cada árbol no se debe cruzar con otro y la suma de los costos de aristas de los árboles es mínima. Un algoritmo de resolución para este problema se puede ver en **[B4]**.

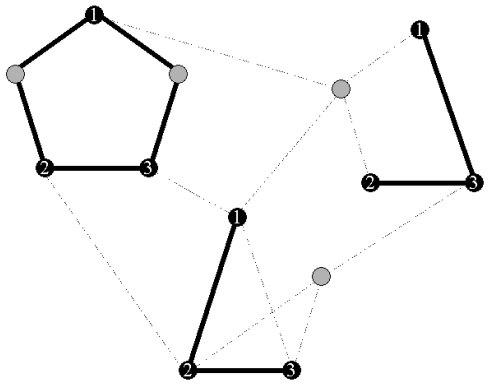


Figura A1.4 Solución óptima del problema del bosque de Steiner en tres componentes.

Problema de Steiner con grado de conectividad más alto

Para problemas de diseño de redes de alta confiabilidad, las soluciones dadas por un modelo del tipo SPN, minimizan el costo total de conexión entre nodos terminales, pero tienen baja confiabilidad si se asume que pueden producirse fallas en las componentes de la red. Si ocurre una falla en un link o nodo, el sistema deja de estar en un estado operativo, lo cual puede ser catastrófico en ciertas aplicaciones.

Al agregar caminos múltiples entre pares de nodos de la red, la confiabilidad de la red aumenta, aumentando también el costo global de la red. Los objetivos: mayor confiabilidad en la red y mínimo costo posible de conexión son contrapuestos; en general, se fija un grado de conexión global a la red, un entero $k \geq 2$, la red diseñada debe poseer al menos k caminos de nodos disjuntos (k caminos de aristas disjuntas) entre todo par de nodos, y tener el menor costo posible.

Problema del subgrafo conexo de Steiner

En ciertos modelos de redes no es necesario que todos los nodos de la red tengan el mismo grado de conexión, sino que se exige que solo un subconjunto de nodos tenga un determinado grado de conexión entre pares de nodos con costo mínimo. El caso más general se conoce como Problema Generalizado de Steiner y se trata especialmente en la siguiente sección. Tres casos particulares son:

- Se requiere que un subconjunto de nodos de la red sea localmente 2-arista-conexo con costo mínimo. (Problema del Subgrafo de Steiner 2-arista-conexo, Steiner 2-edge-connected subgraph problem, STECSP)
- Se requiere que exista un subgrafo 2-arista-conexo de costo mínimo que cubra el subconjunto de nodos especificado. (Problema de la Red de Steiner 2-arista-confiable, Steiner 2-edge-survivable network problem, STESNP)
- Se requiere que un subconjunto de nodos de la red sea localmente k-arista-conexo con costo mínimo. (Problema de la Red de Steiner k-arista-conexo, Steiner k-edge-connected network problem, STESNP). Este es un caso más general que el primero y menor que el GSP.

Toda solución factible del STE2CSP es una solución factible del STE2SNP, además, si los valores de la matriz de costos son estrictamente positivos, la solución óptima del STE2SNP es una solución óptima del STE2CSP. Si $T = V$ ambos problemas coinciden y resulta en un problema del tipo MW2CSN. Los problemas STE2CSP y STE2SNP tienen aplicación en problemas de telecomunicaciones y transporte.

Problema del subgrafo de Steiner 2-arista-conexo (Steiner 2-edge-connected subgraph problem – STE2CSP)

Es el problema de encontrar el subgrafo de G 2-arista-conexo de costo mínimo que cubre el conjunto de nodos terminales T.

Problema del subgrafo de Steiner 2-arista-confiable (Steiner 2-edge-survivable subgraph problem – STE2SNP)

Es el problema de encontrar el subgrafo de G de costo mínimo tal que $i,j \in T / i \neq j$ existen al menos 2 caminos de aristas disjuntas que unen i con j.

Problema del subgrafo de Steiner k-arista-conexo (Steiner k-edge-connected subgraph problem - STEKCNP)

Es el problema de encontrar el subgrafo de G k-arista-conexo de costo mínimo que cubre el conjunto de nodos terminales T.
Una solución a este problema, es solución para el STE2CNP y es un caso especial para el GSP que veremos más adelante.

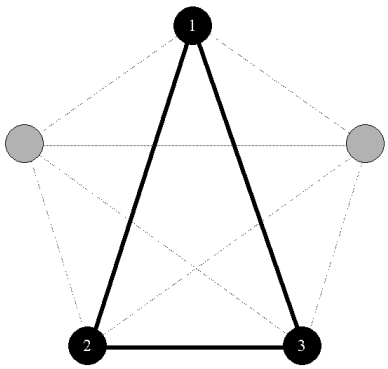
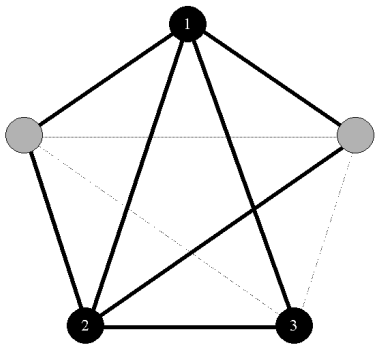


Figura A1.5 Solución óptima del problema del subgrafo de Steiner 2-arista-conexo (y también confiable).

Definición: Un grafo $G = (V, E)$ es llamado k-arista-conexo (k-nodo-conexo), $1 \leq k \leq n-1$ con $n = |V|$, si es conexo y para todo par de nodos $i, j \in V$, hay al menos k caminos de aristas disjuntas (caminos de nodos disjuntos) que los unen.

Problema Generalizado de Steiner (Generalized Steiner Problem - GSP)

El Problema General de Steiner (GSP) formulado por Krarup **[B1]**, es el siguiente. Dado un grafo no dirigido $G=(V,E)$, una matriz de costos c asociados a las aristas, un subconjunto de nodos terminales T, tal que $2 \le n_T \le n$ con $n=|V|$ y $n_T=|T|$, una matriz de dimensión $n_T \times n_T$ $R=\{r_{ij}\}_{i,j \in T}$ cuyos elementos son enteros positivos que indican los requerimientos de conectividad entre todo par de nodos de T, se pretende encontrar un subgrafo G_T de G de costo mínimo tal que para todo par de nodos $i, j \in T, i \neq j$, estos sean localmente r_{ij}-arista-conexos (r_{ij}-nodo-conexos). Ha sido demostrado por Krarup que el problema GSP es NP-Completo tanto para requerimientos de caminos de aristas disjuntas así como de caminos de nodos disjuntos.



$$R:\begin{pmatrix} 0 & 4 & 2 \\ 4 & 0 & 2 \\ 2 & 2 & 0 \end{pmatrix}$$

Figura A1.6 Solución óptima del problema generalizado de Steiner y matriz de requerimientos de conectividad asociada.

La versión arista-conexo del GSP puede ser modelada como un problema de Programación Lineal Entera. Se define un variable binaria x_{ij} para toda arista $(i,j) \in E$. Además, para toda arista $(i,j) \in E$, y $k,l \in T, k \neq l$, se define la variable y_{ij}^{kl} que denota la utilidad de la arista (i,j) en la dirección de i hacia j en un camino que une el nodo terminal k con el nodo terminal l. El siguiente problema de Programación Lineal Entera da la solución óptima al GSP.

Hallar:

$$Min \sum_{(i,j)\in E} c_{ij} \cdot x_{ij}$$

Sujeto a:

$$x_{ij} \geq y_{ij}^{kl} + y_{ji}^{kl} \qquad \forall (i,j) \in E, \forall k,l \in T, k \neq l$$

$$\sum_{(k,j)\in E} y_{kj}^{kl} \geq r_{kl} \qquad \forall k,l \in T, k \neq l$$

$$\sum_{(i,l)\in E} y_{il}^{kl} \geq r_{kl} \qquad \forall k,l \in T, k \neq l$$

$$\sum_{(p,j)\in E} y_{pj}^{kl} - \sum_{(i,p)\in E} y_{ip}^{kl} \geq 0 \qquad \forall k,l \in T, \forall p \in V \setminus \{k,l\}$$

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

$$y_{ij}^{kl} \geq 0 \qquad \forall i,j : (i,j) \in E, \forall k,l \in T, k \neq l$$

Figura A1.7 Formulación del Problema Generalizado de Steiner como un problema de programación lineal entera.

El conjunto de aristas determinado por $\{(i,j) \in E \mid u_{ij} = 1\}$, donde $U = \{u_{ij}\}$ es la solución óptima del problema [PLE_1], define el subgrafo solución G_T. Se puede hacer una formulación análoga para la versión nodo-conexo del GSP, transformando el grafo G mediante la técnica de partición de nodos.

Casos especiales del Problema Generalizado de Steiner [B1]

PROBLEMAS KNCON-KECON

Sea un grafo $G=(V,E)$. Supóngase que a cada nodo $v \in V$ se le asocia un número entero positivo $r(v)$ que representa el "grado de conexión" del nodo v. Se dice que un subgrafo $H=(N,A)$ de G verifica las condiciones de fiabilidad asociadas a los nodos (resp., a las aristas), si para todo par de nodos $s,t \in V$, H contiene al menos $r(s,t)=\min(r(s),r(t))$ caminos de nodos disjuntos (resp., de aristas disjuntas). Al problema de encontrar un subgrafo H que verifique las condiciones de fiabilidad asociadas a los nodos (resp., a las aristas) y que sea de costo mínimo se le denomina kNCON (resp., kECON).

CASOS CON RESOLUCIÓN DE ORDEN POLINOMIAL

Costos idénticos

Cuando los costos de las aristas del grafo $G=(V,E)$ son idénticos, el GSP puede verse como el problema de encontrar un subgrafo con mínimo número de aristas que satisfaga que $\forall i,j \in V$ existan al menos r_{ij} caminos de aristas disjuntas. En algunas variantes, es permitido el uso de aristas múltiples entre pares de nodos.
Couch-Frank presentan un algoritmo de orden polinomial que resuelve el siguiente problema.

Costos 0-1

Si la matriz de costos es una matriz booleana, el problema GSP es conocido como Augmentation Problem. Su formulación es la siguiente.
Dado un grafo $G=(V,E)$, se pretende encontrar el mínimo número de aristas en $V \times V \setminus E$ (posiblemente usando aristas paralelas) tal que al agregarlas al grafo se satisface los requerimientos de conexión (de aristas o nodos) dados por la matriz $R \in Z_+^{V \times V}$.

Costos generales

Cuando los costos de las aristas no son los mismos, existen algoritmos de orden polinomial para el GSP, en casos en los cuales las restricciones de conectividad de aristas (o de nodos) r_i cumplen:

- $r_i = 1$ para todos los nodos i (árbol de cubrimiento de costo mínimo),
- $r_i = k$ para exactamente dos nodos s y t, y $r_i = 0$ para todos los otros nodos i (k-shortest path problem),
- $r_i \in \{0,1\}$ para todos los nodos i, donde o bien el número de nodos del tipo 0 o el número de nodos del tipo 1 esta limitado. Este tipo de problema de Steiner fue resuelto por Lawler.

A1.1.2 Problema de Steiner Geométrico

La variedad geométrica de los problemas de Steiner, expresa los mismos no en un grafo, sino en un sistema de coordenadas geométricas. Des esta manera, los nodos estarán dados por coordenadas geométicas y un posible costo de arista podrá ser la distancia entre los nodos que la determinan.

Problema de Steiner Rectilineo (Rectilinear Steiner problem)

Es un Problema de Steiner representado geométricamente en donde todas las aristas deberán ser horizontales (igual coordenada "y" de los nodos que ldeterminan una arista) o verticales (igual coordenada "x" de los nodos que ldeterminan una arista). Un ejemplo se encuentra en [B3]. Ciertos problemas de diseño de circuitos eléctricos pueden ser modelados de esta manera.

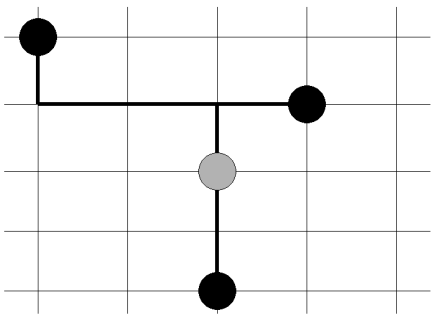


Figura A1.8 Problema de Steiner rectilíneo.

Problema de Steiner sin restricciones

Es el complemento del problema anterior (sin restricciones en la dirección de las aristas)

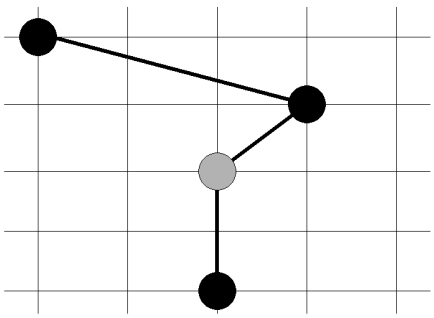


Figura A1.9 Problema de Steiner rectilíneo sin restricciones.

A1.1.3 Algoritmos On-Line y Off-Line

En un problema on-line, los datos de entrada son obtenidos incrementalmente en el tiempo. Un algoritmo on-line toma una secuencia de decisiones, donde cada decision está basada solamente en los datos parciales obtenidos hasta ese momento, sin conocimiento de los datos futuros. Algunos algoritmos on-line toman decisiones tan pronto como obtienen nuevos datos, pero existen otros que obtienen información parcial acerca de sus posibles datos futuros con cierto costo extra. Son usados en numerosas aplicaciones como por ejemplo en problemas de robótica o secuenciamiento de tareas (por ejemplo problemas de ruteo en los cuales nuevos pedidos pueden ser presentados al mismo tiempo que se avanza en el servicio de los anteriores).

Los algoritmos off-line son el estilo más clásico de algoritmos, los cuales en contraposición de los on-line, obtienen todos los datos necesarios antes de tomar alguna decisión.

a) f({d1,d2,d3}) = R

b) f1(d1) = R1
   f2({d1,d2,R1}) = R2
   .
   .
   fN({d1,..,dM,R1,..,RM}) = RN

Figura A1.10 Visualización matemática de las variantes off-line (a) y on-line (b). Las 'R' representan resultados y las 'd' datos.

La variante off-line es la más aplicada y por lo tanto, la mayoría de los algoritmos de resolución del Problema de Steiner se basan en ella. Sin embargo existen soluciones para la variante off-line del Problema Generalizado de Steiner en **[B2]**.

A1.2 Técnicas de Resolución del Problema Generalizado

No existen muchos trabajos que traten este problema, algunos de ellos se pueden encontrar en **[B1,B2,B6..B10]** utilizando algoritmos genéticos como técnica de resolución. A continuación analizaremos algunas de las técnicas propuestas.

A1.2.1 A factor 2 approximation algorithm for the generalized steiner network problem [B6]

Introducción

El artículo plantea el problema de hallar un subgrafo de costo mínimo, de un grafo dado, tal que el número de aristas que cruzan cada corte es al menos cierto requerimiento. Se plantea formalmente el problema de programación entera siguiente:

Sea G = (V,E) un multigrafo no dirigido.
Una función de costo no negativa c: E-> 2+
Una función de requerimientos f::2V -> Z

Hallar: $Min \sum_{e \in E} c_e$

Sujeto a:

$$\forall S \subseteq V : \sum_{e \in \delta_G(S)} x_e \geq f(S)$$

$$\forall e \in E : x_e \in \{0,1\}$$

donde $\delta_G(S)$ denota el conjunto de aristas que tienen exactamente un extremo en S.

Figura A1.11 Problema de programación lineal entera que comprende al Problema Generalizado de Steiner.

El problema generalizado de Steiner es un caso particular de este tomando $f(S) = \max_{i \in S, j \notin S} r_{ij}$. Se sabe además que f es propia y débilmente supermodular (toda función propia es débilmente supermodular), lo cual es una propiedad fundamental para la solución.

El algoritmo es de la clase de algoritmos de redondeo, los cuales usan una solución fraccional óptima para obtener una buena solución entera.
Se introduce la idea de redondeo iterativo, el cual busca una solución fraccional óptima, x, en la cual por lo menos una arista e tiene xe de por lo menos ½. Ese xe puede ser redondeado a 1 doblando su contribución al costo de la solución. La parte de los requerimientos de corte que no son satisfechos por las aristas integralmente elegidas, definen un problema residual; el cual puede ser modelado por el problema de programación entera definido arriba.

Definición: **Una función** $f : 2^V \rightarrow Z_+$**, es *propia,* si** $f(V) = 0$ **y se cumplen las siguientes condiciones:**

1. **Para cada subconjunto S de V,** $f(S) = f(V - S)$
2. Para todo par de subconjuntos disjuntos A y B, $f(A \cup B) \leq \max\{f(A), f(B)\}$

Solución

La idea entonces, es primeramente encontrar una solución fraccional al problema LP (presentado abajo). Este problema (LP), no es más que la relajación lineal del problema planteado al comienzo:

Sea G = (V,E) un multigrafo no dirigido.

Una función de costo no negativa c: E-> 2+

Una función de requerimientos f::2V -> Z, débilmente supermodular.

Hallar: $Min \sum_{e \in E(G)} c_e x_e$

Sujeto a:

$$\forall S \subset V : \sum_{e \in \delta_G(S)} x_e \geq f(S)$$

$$\forall e \in E(G) : 1 \geq x_e \geq 0$$

Figura A1.12 Relajación del problema de la figura 4.1

Encontrada esta solución básica óptima, se incluyen en la solución, todas las aristas con valores ½ o más. Se borrar las aristas, que fueron incluidas en la solución, del grafo y se resuelve el problema residual.

Conclusión

El proceso obtiene una solución aproximada de un factor de 2 del costo de la solución óptima, asumiendo que puede hallar una solución óptima fraccional del problema residual. El método no explota el máximo poder de la programación lineal. Partiendo de una solución fraccional, redondeando esta, no se obtiene la mejor solución para continuar con el proceso circular.

A1.2.2 When trees collide: An approximation algorithm for the generalized Steiner problem in Networks [B1,B7]

Introducción

Este artículo presenta una solución al GSP mediante la definición de un problema equivalente denominado Multiterminal Network Synthesis (MNS). Aquí se propone un escenario real, la conexión de Centrales Hidroeléctricas entre un grupo de países limítrofes para satisfacer sus requerimientos energéticos, y en base a este escenario se formaliza el modelo MNS.

Formulación

Sea $G = (V, E)$ un grafo no dirigido con costos no negativos asociados a sus aristas, y sea R un conjunto de pares de nodos $R = \{(o_i, d_i), i \in 1..k, o_i, d_i \in V\}$, donde a los nodos o_i y d_i se los denomina sitios, las parejas (o_i, d_i) pares de sitios, y k es la cantidad de pares de sitios en R. Un subgrafo H de G, se le llama Requirement Join si para todo par de nodos $(o_i, d_i) \in R$ el subgrafo H contiene un camino que une los sitios o_i y d_i. A los pares de nodos de R se les denomina requerimientos de conexión, pues ellos representan las restricciones de conexión que deben ser satisfechas por el subgrafo Requirement Join. Un Requirement Join asociado a un conjunto de requerimientos de conexión R, se lo denota como $R - join$.

Dado un grafo $G = (V, E)$ el problema de hallar el $R - join$ de costo mínimo tiene múltiples aplicaciones prácticas. Por ejemplo, el problema anteriormente descripto. Un caso particular de éste problema, es el Steiner Tree Problem in Networks (SPN).
Supóngase que al problema descripto se le agrega la restricción en la cual los vértices que deben estar conectados, lo deben estar por un cierto número de caminos múltiples disjuntos. Esto contribuye al diseño de una red de conexión más confiable, haciéndola menos vulnerable a fallas en los "links".

Se define un R-multijoin como un subgrafo de G cuyo conjunto de aristas determinan al menos ri caminos de aristas disjuntas entre los sitios oi y di , para todo requerimiento de conexión (oi,di,ri). El costo de un R-multijoin es la suma de los costos de las aristas que lo componen, contando sus multiplicidades.

El problema de encontrar un R-multijoin de costo mínimo, es el MNS. El MNS sin aristas múltiples es equivalente al GSP formulado por Krarup, la transformación de un problema a otro es trivial.

Previo al algoritmo aproximado presentado por Agrawal-Klein-Ravi para resolver el problema de encontrar el $R - multijoin$ de mínimo costo de un grafo, no se conocía ningún otro algoritmo aproximado para resolver el GSP. El algoritmo que ellos presentan es aplicable aún cuando hay aristas múltiples entre pares de nodos. Una descripción completa de este algoritmo se encuentra en **[B7]**.

Solución

La idea del algoritmo es hacer crecer árboles de búsqueda en amplitud desde los nodos, cada árbol dedicado a solucionar el problema de conectividad en ese nodo inicial. El algoritmo utiliza la noción de timestep (paso de tiempo). A cada timestep, cada árbol crece un nivel adicional. Cada árbol crece hasta que todos los sitios que contiene hallan encontrado sus parejas. Cuando los árboles colisionan, son unidos. Mientras que el algoritmo hace crecer los árboles, construye redes de cubrimiento sobre los sitios en cada árbol, y luego que el proceso termina, se tendrá una solución al problema generalizado.

Conclusión

La solución propuesta por Agrawal-Klein-Ravi en **[B7]**, trae consigo el siguiente resultado: Existe un algoritmo de orden polinomial que encuentra un R-multijoin de costo a lo sumo un factor $(2-2/k)\cdot\lceil \log_2(r_{\max}+1)\rceil$ veces el costo de la solución óptima, donde rmax es el valor de requerimiento de conexión más grande y k es el número de sitios en R.

A1.2.3 Algoritmo Backtracking Exacto [B1]

Introducción

Dado un grafo $G=(V,E)$ en las condiciones del GSP, se denota al espacio de soluciones factibles de la instancia del problema como Γ_{GSP}. El mismo viene dado por el conjunto de subgrafos de G que satisfacen la matriz de requerimientos R; o sea $G_s(N,A)\in\Gamma_{GSP}$ si $T\subseteq N\subseteq V$, $A\subseteq E$ y además $\forall i,j\in T$ existen r_{ij} caminos de aristas disjuntas en $G_s(N,A)$.

El algoritmo propuesto, consta de dos procedimientos mutuamente recursivos, los cuales analizan el espacio de soluciones factibles Γ_{GSP} del GSP en búsqueda de la solución óptima, descartando mediante la aplicación de funciones de cota, ramas del árbol de soluciones correspondiente a la instancia del problema.

Solución

Se define el conjunto de aristas $A_{sol}\subseteq E$, el conjunto de nodos $N_{sol}\subseteq V$, y la variable min_costo (inicializada en INF) como globales al algoritmo y es donde se almacena la mejor solución encontrada hasta el momento por el algoritmo. También se asume como globales al algoritmo las representaciones del grafo $G=(V,E)$ original, la matriz de costos c asociada a las aristas, la matriz de requerimientos de conexión R, y el conjunto de nodos terminales T.

El algoritmo consta de dos partes: un procedimiento denominado "ALG_G_STEINER" y un segundo procedimiento auxiliar denominado "ALG_G_A". Los pseudocódigos de dichos procedimientos son los siguientes.

```
Procedure ALG_G_STEINER(A,N,i,costo,num_term);
Begin
1  OK=ACT_SOL(A,N,costo,num_term);
2  if (i≤n) and (costo<min_costo) and (not(OK)) then
3    if (i∉T) then
4      ALG_G_STEINER(A,N,i+1,costo,num_term);
5      if Test_Nodo_Steiner(i,N,A) then
6        ALG_G_A(A,N∪{i},i,1,costo,num_term,0,2);
7      endif;
8    else
9      r_max_c=MaxReqCon(i);
10     if Test_Nodo_Terminal(i,N,A) then
11       ALG_G_A(A,N∪{i},i,1,costo,num_term+1,0,r_max_c);
12     endif;
13   endif;
14 endif;
End;
```

Figura A1.13 Procedimiento para hallar la solución el Problema Generalizado de Steiner utilizando backtracking.

```
Procedure ALG_G_A(A,N,i,j,costo,num_term,cant_a,r);
Begin
1  ady_may=AdyMayores(i,j);
2  c_min=MinRestoAdy(i,j);
3  resto_r=MAX(r-cant_a,0);
4  if (cant_a+ady_may≥r) and (costo+(resto_r·c_min))<min_costo)
then
5    if (j< i) then
6      if (j∈N) and ((i,j)∈G) then
7        ALG_G_A(A∪{(i,j)},N,i,j+1,costo+c_ij,num_term,cant_a+1,r);
8        ALG_G_A(A,N,i,j+1,costo,num_term,cant_a,r);
9      else
10       ALG_G_A(A,N,i,j+1,costo,num_term,cant_a,r);
11     endif;
12   else
13     ALG_G_STEINER(A,N,i+1,costo,num_term);
14   endif;
15 endif;
End;
```

Figura A1.14 Procedimiento mutuamente recursivo al de la figura 4.3 utilizado en la resolución.

Se utilizan además las siguientes funciones auxiliares:

ACT_SOL: Indica si se tiene una mejor solución factible que la solución actual.

Test_Nodo_Steiner: Recibe como parámetros de entrada un nodo de Steiner $i \in V \setminus T$, un conjunto de aristas $A \subseteq E$, y un conjunto de nodos $N \subseteq V$. Devuelve TRUE si el nodo de Steiner i puede llegar a formar parte de una solución factible del espacio Γ_{GSP} con menor costo que la actual solución, construida a partir de $G_s(N, A)$, agregando nodos que están en el conjunto $V^{(\geq i)} = \{v \in V / v \geq i\}$ y aristas de $(E \setminus E^{(<i)})$, donde $E^{(<i)} = \{(u,v) \in E / u,v < i\}$; devuelve FALSE en caso contrario.

Test_Nodo_Terminal: Recibe como parámetros de entrada un nodo terminal $i \in T$, un conjunto de aristas $A \subseteq E$, y un conjunto de nodos $N \subseteq V$. Devuelve TRUE si existe la posibilidad de encontrar una mejor solución factible en el espacio Γ_{GSP}, formada a partir del subgrafo $G_s(N \cup \{i\}, A)$ agregando nodos del conjunto $V^{(>i)} = \{v \in V / v > i\}$ y aristas de $(E \setminus E^{(<i)})$; devuelve FALSE en caso contrario. Al igual que "Test_Nodo_Steiner", los test realizados por la función "Test_Nodo_Terminal" se darán en Proposición 10.

MaxReqCon: La función "MaxReqCon" recibe como entrada un nodo $i \in T$, y devuelve el mayor requerimiento de conexión en R, que tiene como extremo al nodo terminal i. Es decir, se calcula el valor $Max\{r_{ij}, \forall j \in T\}$.

AdyMayores: Devuelve los nodos adyacentes a un vértice, mayores, en el orden en que se encuentran en la representación matricial del grafo.

MinRestoAdy: Esta función recibe como parámetros de entrada dos nodos $i, j \in V$, y devuelve el menor costo de una arista $(i,k) \in E$ tal que $k \geq j$.

A1.2.4 Algoritmo Aproximado Basado en Ant System [B1]

Introducción

La meta heurística conocida como Ant Systems se basa en el proceso de búsqueda de alimentos de la especie de hormiga conocida como Linepithema Humile.

El comportamiento es el siguiente: se tiene un conjunto de hormigas, las que van saliendo del hormiguero de a una, la hormiga explora el área que la rodea eligiendo una dirección totalmente aleatoria. Si existe un rastro (dejado por una hormiga anterior) presente, entonces la hormiga elige la dirección del rastro con alta probabilidad. Si existen dos o más rastros diferentes, la hormiga elige con mayor probabilidad el rastro más "fuerte" (aquel por el que hayan pasado más hormigas). El proceso es un proceso de retroalimentación positiva, pues el rastro es reforzado en esa dirección con el paso de las hormigas. Otra característica del fenómeno, es que el rastro tiende a desaparecer si no es utilizado gradualmente.

Los resultados teóricos de esta metodología, postulan que las hormigas rápidamente elegirán el camino más corto que las conduzca a la fuente de alimentos, debido que será este el que contenga el rastro más fuerte.

Se adaptó esta metodología para la resolución del Problema Generalizado de Steiner, lo cual trataremos a continuación.

Algoritmo

El algoritmo recibe como entradas:

- un grafo simple $G = (V, E)$,
- el conjunto de nodos terminales $T \subseteq V$,
- matriz de costos asociadas a las aristas c,
- matriz de requerimientos de conexión entre nodos terminales R.

En base a estos parámetros, el algoritmo intenta encontrar una "buena" solución factible dentro del espacio Γ_{GSP}. La idea de su funcionamiento es la siguiente. Mientras no se cumpla un cierto número máximo de iteraciones (definido de antemano) y no se haya encontrado una cierta cantidad de soluciones factibles (también definida de antemano), el algoritmo realiza lo siguiente. Se inicializa una serie de parámetros, entre ellos la solución actual con $N = T$, $A = \varnothing$ y $\cos to\_sol = INF$. Luego, $\forall i, j \in T$ se trata de encontrar una cantidad r_{ij} de caminos de aristas disjuntas que unan ambos nodos terminales, construyéndose de esta manera una solución factible para la instancia del GSP.

La determinación de cada uno de estos caminos, se hace utilizando un algoritmo llamado "Busco_Camino", el cual determina caminos entre pares de nodos terminales sobre un subgrafo de G, y selecciona uno de ellos de acuerdo a un cierto criterio. A grandes rasgos, el funcionamiento del algoritmo "Busco_Camino" es el siguiente. Sea $G_s(V, E \setminus A)$ el subgrafo de G sobre el cual se desea determinar un camino que una los nodos terminales i y j; para ello se ubica un cierto número de hormigas en el nodo terminal i y en el nodo j (la cantidad de hormigas a ubicar en cada nodo depende de r_{ij} y del grado del nodo en el subgrafo en cuestión), luego las hormigas se "mueven" sobre el subgrafo encontrándose una cierta cantidad de caminos que unen ambos nodos terminales.

Sobre el conjunto de caminos que unen i con j encontrados por las hormigas, se selecciona aquél que satisface el denominado "Criterio de Selección de Camino" (dicho criterio se define luego formalmente). Una vez seleccionado un camino $\overline{w}$ por el algoritmo "Busco_Camino", se actualiza la solución actual mediante las asignaciones $N = N \cup Nodos(\overline{w})$, $i \in V$ y $costo\_sol = costo\_sol + COSTO(\overline{w})$.
Cada vez que se agrega un nuevo camino a la solución actual, se controla que el costo de ésta sea menor que el de la mejor solución factible encontrada hasta el momento, de no ser así, el algoritmo "GSP_Ant_System" intenta construir una nueva solución factible desde el comienzo. De igual manera, si para algún par de nodos terminales $i, j \in T$ el algoritmo "GSP_Ant_System" no puede determinar mediante múltiples aplicaciones del algoritmo "Busco_Camino" la cantidad r_{ij} de caminos de aristas disjuntas requerido, entonces el algoritmo resetea los parámetros e intenta construir una nueva solución factible.

Si $\forall i, j \in T$ se logran encontrar r_{ij} caminos de aristas disjuntas, construyendo una solución $G_s(N, A)$ con costo menor que el de la mejor solución factible encontrada hasta el momento, entonces, el algoritmo "GSP_Ant_System" actualiza la mejor solución mediante las asignaciones $N_{Sol} = N$, $A_{Sol} = A$ y $min\_costo = COSTO(A)$.

El algoritmo "GSP_Ant_System" retorna la mejor solución factible encontrada luego de un cierto número de iteraciones. En caso de no haberse hallado ninguna solución factible se indica mediante un parámetro booleano que la búsqueda no fue exitosa.
Seguidamente se brinda la definición del "Criterio de Selección de Camino" utilizado por el procedimiento "Busco_Camino".

Criterio de Selección de Camino

Sea $W^{i-j} = \{w^{i-j}\}$ el conjunto de caminos que unen el nodo terminal i con en el nodo terminal j, determinado por un conjunto de hormigas en una instancia de ejecución del algoritmo "Busco_Camino". El camino seleccionado, es aquel $\overline{w} \in W^{i-j}$ que satisface

$$\frac{COSTO(\overline{w})}{NUM\_TERM(\overline{w})+2} = Min\left\{\frac{COSTO(w^{i-j})}{NUM\_TERM(w^{i-j})+2}, w^{i-j} \in W^{i-j}\right\}.$$

Donde $COSTO(w)$ es el costo del camino w y $NUM\_TERM(w)$ es el número de terminales distintos de i y j presentes en w y para los cuales todavía existen requerimientos de conexión no satisfechos en la solución actual. La idea es seleccionar el camino que une los nodos terminales i y j que halla recorrido la mayor cantidad de nodos terminales con el menor costo posible; los nodos terminales tomados en cuenta son aquellos para los cuales su requerimiento de conexión con algún otro nodo terminal todavía no ha sido satisfecho por la solución actual (en fase de construcción).

Aplicando este criterio, sean $w_1^{i-j}, w_2^{i-j} \in W^{i-j}$, entonces se cumple:

- si $COSTO(w_1^{i-j}) = COSTO(w_2^{i-j})$ y $NUM\_TERM(w_1^{i-j}) > NUM\_TERM(w_2^{i-j})$, entonces es preferido w_1^{i-j} antes que w_2^{i-j} (a igual costo se prefiere el camino que haya recorrido mayor número de terminales),
- si $COSTO(w_1^{i-j}) < COSTO(w_2^{i-j})$ y $NUM\_TERM(w_1^{i-j}) = NUM\_TERM(w_2^{i-j})$, entonces es preferido w_1^{i-j} antes que w_2^{i-j} (a igual número de terminales recorridos, se prefiere el camino de menor costo).

Pseudocódigo del algoritmo

El siguiente es el pseudocódigo detallado del algoritmo "GSP_Ant_System".

```
Algoritmo GSP_Ant_System(in V:Nodos, E:Aristas, T:Nodos, c:Costos ,
                         R:Matriz_Req_Con;  out N_Sol:Nodos,  A_Sol:Aristas,
exito:boolean);
cant_iter=0;
cant_sol_fact=0;
min_costo=INF;
Leer_Ant_System_Param(param);
while ((cant_iter≤MAX_ITER) and (cant_sol_fact<MAX_SOL_FACT)) do
   Inicializo_1(T,R,N,A,M,B,costo_sol);
  OK=TRUE;
   for each i, j ∈ T / i ≠ j  do
       while (M_ij>0) and (OK=TRUE) do
          Grafo_Cociente(E,A,A_c);
          Busco_Camino(V,A_c,R,c,param,i,j,p,fallo);
           if (fallo=FALSE) then
               Actualizar_Solucion(c,p,N,A,costo_sol);
               if (costo_sol<min_costo) then
                   Actualizar_Matrices(p,T,R,M,B);
               else  OK=FALSE;
               end_if;
           else  OK=FALSE;
           end_if;
       end_while;
       if (OK=FALSE) then
           exit_for_each();
       end_if;
   end_for_each;
   if (OK=TRUE) then
       Actualizar_Mejor_Solucion(N,A,costo_sol,N_Sol,A_Sol,min_costo);
       cant_sol_fact=cant_sol_fact+1;
       cant_iter=0;
   else  cant_iter=cant_iter+1;
   end_if;
end_while;
if  (cant_sol_fact>0) then
   exito=TRUE;
else
   exito=FALSE;
end_if;
```

Conclusiones

Sea $G = (V, E)$ un grafo en las condiciones del GSP. Sean además: $n = |V|$ la cantidad de nodos del grafo, $n_T = |T|$ la cantidad de nodos terminales, $r_{Max} = Max\{r_{ij}; i, j \in T\}$ el mayor requerimiento de conexión entre pares de nodos terminales y MAX_ANTS una cota máxima a la cantidad de "hormigas" posibles a ubicar en el grafo G.
El orden del algoritmo "GSP_Ant_System" viene dado entonces por:
$O\left(MAX\_ITER \times MAX\_SOL\_FACT \times r_{Max} \times (T_{B\_C} + n(n-1) + {n_T}^2 + n_T)\right)$, donde:
$T_{B\_C} = MAX\_INTENTOS \times MAX\_CAM \times n \times (MAX\_ANTS + n \cdot (n-1))$ es la cantidad de operaciones fundamentales del algoritmo de determinación de caminos en el peor caso.

Los problemas de validación fueron seleccionados pensando en someter al algoritmo "GSP_Ant_System" a instancias del problema GSP donde las topologías son pequeñas y donde además surgen diferentes situaciones particulares que ponen a prueba la capacidad del algoritmo de armar una "buena" solución en su fase de construcción de una solución factible. En todos los casos, el algoritmo tuvo un muy buen desempeño obteniéndose la solución óptima en todos los problemas.

Los resultados numéricos obtenidos en las pruebas experimentales, revelan un buen desempeño del algoritmo "GSP_Ant_System", luego de haberse realizado una minuciosa depuración de los diferentes juegos de valores de los parámetros. De cinco problemas, se alcanzó la solución óptima en los 4 primeros. En el Problema 5 se obtuvo en diferentes corridas (con parámetros distintos) dos soluciones factibles con una diferencia porcentual de costo inferior al 5% con respecto al costo óptimo.

Estos resultados son muy buenos si se toma en cuenta que el cálculo de la solución óptima es un problema NP-difícil, donde los órdenes de los algoritmos que hallan la solución óptima son exponenciales en la cantidad de nodos terminales y aristas.

A1.2.5 A Primal-Dual approximation algorithm for Generalized Steiner Network problems [B8]

No se tuvo acceso a este artículo, sin embargo es importante destacar su existencia por ser de los pocos que tratan este problema.

Apéndice 2 Algoritmos Genéticos

Dado el espacio de todas las posibles soluciones a un problema, y partiendo de una solución inicial, podemos ser capaces de hallar una mejor solución o la única abordando el problema desde varios perfiles. Si esta búsqueda se basa en una función indicatriz de lo buena que es una solución, o que tan cerca se está de esta, el problema se transforma en un problema de optimización. Cualquiera de estos casos pueden ser abordados desde varios perfiles diferentes, tales como algoritmos iterativos, heurísticas, etc.

Los algoritmos genéticos son métodos sistemáticos para la resolución de este tipo de problemas (búsqueda y optimización) desde una perspectiva muy particular, la imitación del proceso de evolución de los organismos biológicos.

Como la evolución, estos operan con un conjunto de individuos que representan potenciales soluciones para un problema dado. Utilizando el concepto de "supervivencia del más apto", se intenta descartar los malos y producir mejores (mejor adaptados) individuos (soluciones) mediante la aplicación iterativa de este proceso de "evolución". Generalmente no se intenta encontrar una solución exacta sino aproximar lo mejor posible a esta.

El poder de esta técnica se basa en la robustez de la misma y que puede manejarse de forma exitosa en una gran gama de problemas, incluso en aquellos en los cuales se le hace difícil a otros métodos el hallar una solución **[B13]**. Sin embargo, no existe una basta teoría que sustente la correctitud y viabilidad en el uso de esta técnica. Principalmente, demostraciones empíricas han hecho factible el uso de esta técnica.

El término "genético" deriva del hecho de que los individuos son representados en base a su estructura genética, lo cual está tratado con mayor profundidad en la sección 1. La segunda sección, muestra una breve comparación entre esta técnica y otras utilizadas para la resolución del mismo tipo de problemas. Por último, la sección 3 plantea algunos conceptos avanzados de esta técnica.

A2.1 Conceptos Básicos

Tal como se introdujo, los algoritmos genéticos intentan imitar el proceso natural de evolución de los organismos biológicos. Para ello se considera que nuestro universo es el espacio de soluciones factibles del problema a tratar, en él existe un conjunto de individuos los cuales representan posibles soluciones al problema o aproximaciones a esta. A cada individuo se le asigna un valor de aptitud (fitness), el cual indica que tan buena solución es. Los individuos con mejor fitness tienen más oportunidades de "reproducirse" con otros individuos (propagar sus propiedades genéticas para generar mejores soluciones). La "reproducción" entre dos individuos produce nuevos con características heredadas de sus "padres". Los menos aptos serán los más propicios a desaparecer. Esta nueva generación contiene una mejor proporción de las características de los buenos individuos y en consecuencia, puede llevar consigo mejores resultados.

A2.1.1 Codificación

El funcionamiento de un algoritmo genético simple, tal como lo hemos descrito hasta el momento, está representado en el siguiente seudo código.

```
begin
  pob := generar_población_inicial();
   /*Genero una población inicial de posibles soluciones
     al problema o aproximaciones a esta*/
  pob := calcular_fitness(Pob);
   /*Calculo el valor de aptitud de cada individuo*/

  while (not salir) do
    for 1 to largo(Pob) /2 do
      /*Tomo solo la mitad de los individuos para
        reproducir*/
      (ind1,ind2) := selecciono_dos_individuos(Pob);
      /*Selecciono dos individuos según su aptitud. Un
        mismo individuo puede seleccionarse dos veces*/
      (n_ind1,n_ind2) := cruza(ind1,ind2);
      /*Efectúo la "reproducción" generando dos nuevos
         individuos*/
      (n_ind1,n_ind2) := mutación(n_ind1,n_ind2);
      /*Efectúo la "mutación"*/
      (n_ind1,n_ind2) := calculo_aptitud(n_ind1,n_ind2);
      n_pob := n_pob + n_ind1 + n_ind2;
      /*La nueva población se forma a partir de los
        nuevos individuos.*/
    end for
    pob = n_pob;
    if (se cumple la condición de parada) then
      /*La condición de parada puede ser la convergencia
        del algoritmo a la solución esperada, la
        conclusión de un quantum de tiempo, etc*/
      salir = true;
    end if
  end while
end
```

A2.1.2 Representación Genética

Toda solución debe ser representada como un conjunto de parámetros, estos (conocidos como genes) se unen para formar una posible solución (llamado cromosoma). La representación del cromosoma es uno de los factores determinantes del buen o mal funcionamiento del algoritmo. Cada gen del cromosoma posee un valor particular (alelo) y una posición dentro de este (locus). Se ha demostrado que lo ideal es el uso de un alfabeto binario para representar los diferentes valores de un locus, aunque es posible la utilización de otras representaciones, tal como se verá en el capítulo de conceptos avanzados. En términos genéticos, el conjunto de parámetros representados por un cromosoma es denominado genotipo, este, contiene toda la información para representar un organismo (denominado fenotipo).

De aquí en adelante trabajaremos con una representación binaria, sin pérdida de generalidad.

Se introducirá como ejemplo, maximizar la función $f(x) = x^2$ en el intervalo entero [0,31]. Se utiliza una representación binaria para representar los valores de x, y la función de fitness es la evaluación de la función f(x) en esos valores.

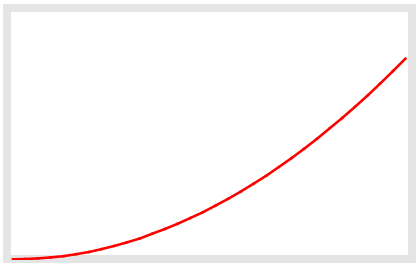


Figura A2.1 Función $f(x) = x^2$ en [0,31]

A2.1.3 Funcion de Fitness

Durante la evaluación de un individuo, se le asigna a este una puntuación (fitness) que determina lo cerca que este está de la mejor solución. La función de fitness mapea cada posible solución en el espacio de soluciones con un valor específico. La misma depende estrictamente del problema a tratar, por ejemplo: al tratar de maximizar una función, la función de fitness podría ser la evaluación de esa función con los valores del cromosoma.

Number	String	Fitness	% of the Total
1	01101	169	14.4
2	11000	576	49.2
3	01000	64	5.5
4	10011	361	30.9
Total		1170	100.0

Figura A2.2 Población inicial del ejemplo con sus valores de fitness asociados y el porcentaje del total del fitness asociado a cada individuo.

A2.1.4 Selección

La función de fitness determinará que individuos serán los que se deban procrear y cuales perecerán. La selección (selection) de estos individuos se realiza generalmente de alguna de las siguientes formas (no son las únicas):

- Estado estacionario: en cada etapa se mantiene un porcentaje de la población para la siguiente generación. Se ordena la población por orden de fitness y los N mejor dotados son seleccionados para la reproducción. El resto es eliminado.
- Ruleta: se crea una ruleta en la cual cada individuo tiene asignada una porción proporcional a su aptitud. Se eligen aleatoriamente los individuos y debido a que los más aptos tienen un área mayor que los menos aptos, probablemente serán seleccionados más veces.
- Torneo: se van tomando grupos de individuos de la población que compiten entre ellos, los ganadores son los que van a ser seleccionados.

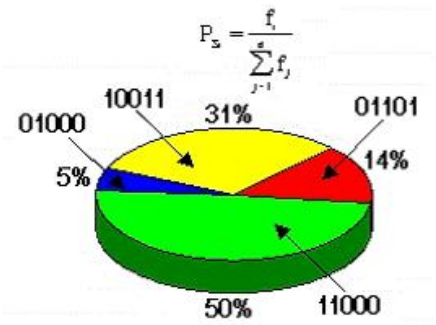


Figura A2.3 Ruleta creada para la población inicial del ejemplo.

A2.1.5 Cruza

Luego de elegidos los individuos que se deberán reproducir, es necesario combinar su información genética para la construcción de los nuevos individuos. Este proceso es denominado cruza (crossover). La cruza no se debe realizar siempre, sino con cierta probabilidad (generalmente entre 0.6 y 1.0), de forma que un individuo de buena aptitud pueda pasar a la nueva generación sin necesidad de modificar su información genética.

La cruza se puede realizar de varias formas, algunas de estas son generales para todo tipo de representación y otras dependen exclusivamente del problema específico. Explicaremos a continuación las cruzas más generales y usadas:

A2.1.5.1 Cruza de n puntos

La cruza de n puntos (n-point crossover), se basa en la elección aleatoria de n posiciones de los cromosomas (llamados puntos de corte). La información genética de las partes se combina intercaladamente para formar los dos nuevos individuos.
Las cruzas más comunes son las de uno y dos puntos.

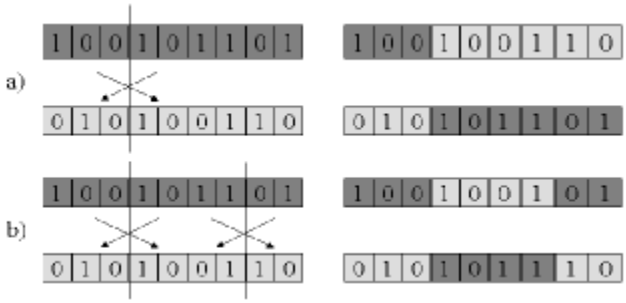


Figura A2.4 Cruza de uno (a) y dos puntos (b)

A2.1.5.2 Cruza uniforme

En la cruza uniforme (uniform crossover), se intercambia con cierta probabilidad uno a uno de los genes de los cromosomas para formar los dos nuevos individuos.

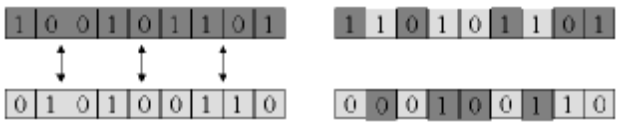


Figura A2.5 Cruza uniforme

No han aparecido pruebas que fundamenten la preferencia en la utilización de alguna de las anteriores cruzas.

La reproducción de individuos puede generar que estos no sean factibles, es decir, no representen una solución correcta, por lo que se deberán agregar métodos para corregir estas desviaciones. Algo que puede llegar a optimizar el funcionamiento del algoritmo genético es la realización de cruzas que certifiquen la correctitud de los nuevos individuos.

Padres	Hijos
01|101(169)	01000(64)
11|000(576)	11101(841)

Figura A2.6 Cruza de un punto para dos individuos del ejemplo.

A2.1.6 Mutacion (mutation)

La mutación (mutation) es aplicada a cada individuo luego de la cruza. Esta altera aleatoriamente cada gen con una muy baja probabilidad (generalmente 0.001). En definitiva, la cruza es la más importante de las dos técnicas para una rápida exploración del espacio de búsqueda. La mutación provee una pequeña porción de búsqueda aleatoria, y ayuda a asegurarse de que ningún punto en el espacio de búsqueda tenga una probabilidad nula de ser considerado. Es más conveniente usar otros mecanismos de generación de diversidad, como aumentar el tamaño de la población, o garantizar la aleatoriedad en la población inicial.

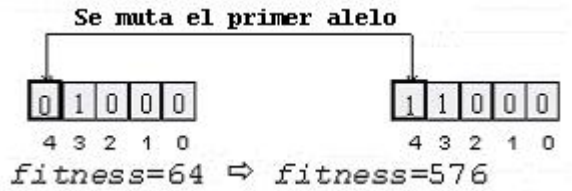


Figura A2.7 Mutación de uno de los nuevos individuos de la población.

A2.1.7 Convergencia

Si el algoritmo genético está correctamente implementado, la población evolucionará durante sucesivas generaciones de tal forma que el valor de aptitud del mejor individuo y del promedio crezcan hacia el óptimo global. Un problema a considerar es la posibilidad de que un grupo de individuos con aptitud alta pero no óptima dominen rápidamente la población haciendo que el método converga a un óptimo local. Por otro lado puede ocurrir que la convergencia se realice muy lentamente. En cualquiera de los dos casos la solución es mejorar el método de selección de individuos de la población.

La condición de parada de un algoritmo genético no tiene porqué depender estrictamente de la convergencia de los resultados, sino también de la finalización de un quantum de tiempo, la llegada a una generación determinada, etc.

A2.1.8 Parámetros genéticos

Es importante analizar de que manera algunos parámetros influyen en el comportamiento de los algoritmos genéticos, para poder determinarlos acorde a las necesidades del problema y de los recursos disponibles.

Tamaño de la población: afecta en el desempeño global y la eficiencia del algoritmo. Con una población pequeña, el desempeño puede caer, ya que abarca solamente una pequeña parte en el espacio de búsqueda de una sola vez. Una gran población cubre una mayor "área" de incidencia y puede evitar la rápida convergencia a mínimos locales. Sin embargo, para esto es necesario de mayores recursos computacionales y que el algoritmo trabaje por más tiempo.

Tasa de cruzamiento: Cuanto mayor es esta taza, más rápidamente se introducen nuevos individuos a la población. Pero, con esta taza, individuos de gran aptitud pueden ser intercambiados más rápidamente. Con un valor bajo, el algoritmo se puede tornar muy lento. Generalmente, se utilizan valores cercanos a 0,6.

Taza de mutación: Una taza baja provee mayor diversidad en la población, evitando estancamientos en esta. El incremento de esta taza logra un incremento en la aleatoriedad del algoritmo. Generalmente, se utilizan valores cercanos a 0,001.

Otras tazas: Otro tipo de tazas, tal como la frecuencia de aplicación de operadores que introducen diversidad, provocan la aleatorización del algoritmo si son usadas con grandes valores.

A2.2 Comparación con Otras Técnicas

La elección de los algoritmos genéticos como técnica de optimización y búsqueda, se basa en la robustez que estos proveen y en la gran cantidad de áreas de problemas en los cuales pueden ser aplicados, incluyendo aquellas en las cuales es difícil hallar una solución con otros métodos **[B13]**.

A continuación se muestra una evaluación de los algoritmos genéticos haciendo énfasis en las ventajas y desventajas de estos comparativamente con otras técnicas usadas para resolver la misma clase de problemas **[B15]**.

A2.2.1 Búsqueda aleatoria

Es raramente usada debido a la inseguridad que nos provee.

Un algoritmo genético es una especie de búsqueda aleatoria controlada e iterativa.

A2.2.2 Métodos de gradiente

Existen métodos que se basan en el gradiente de la función para guiar la dirección de la búsqueda. Si la derivada de la función no puede ser computada, porque es discontinua, por ejemplo, estos métodos fallan.

Algunos métodos se llaman de escalado (hillclimbing), comienzan desde un punto "escalando" hasta el máximo. Funcionan bien si existe solo un pico pero pueden fallar con varios. Si el primer pico encontrado es un mínimo local, el método no sigue.

Los algoritmos genéticos no tienen estos problemas ya que no dependen de particularidades en las funciones a optimizar, ni de la localidad de los óptimos.

A2.2.3 Búsqueda Iterativa

La búsqueda aleatoria y los métodos de gradiente pueden ser combinados para obtener una búsqueda de escalada iterativa. Una vez que un pico es alcanzado, se comienza de nuevo con otro punto aleatorio. Esta técnica tiene la ventaja de la simplicidad, y funciona bien si la función no tiene muchos máximos locales. Sin embargo estos métodos no guardan una imagen del camino ya recorrido, por lo que sus intentos se basan tanto en puntos con alto *fitness* como con bajo.

Un algoritmo genético, en comparación, comienza con una población aleatoria, y fija sus intentos en regiones con alto *fitness*. Esto es una desventaja si el máximo se encuentra en una pequeña región rodeada de regiones con bajo fitness. Este tipo de funciones son difíciles de optimizar por cualquier método, y en este caso la simplicidad de la búsqueda iterativa es preferida.

Una búsqueda iterativa puede asegurar la convergencia, mientras que los algoritmos genéticos, pueden tardar mucho en converger, o no hacerlo en absoluto, dependiendo de cierta medida de los parámetros utilizados (tamaño de la población, número de generaciones, etc).

A2.2.4 Simulated Annealing

Es esencialmente una versión modificada de hillclimbing. Comenzando desde un punto aleatorio en el espacio de búsqueda, se realiza un movimiento aleatorio. Si el movimiento lleva a un punto más alto, se lo acepta, sino, es aceptado solamente con probabilidad *p(t),* donde *t* es el tiempo. La función *p(t)* comienza cerca de 1, pero gradualmente se reduce a cero.

Al igual que la búsqueda aleatoria se maneja solamente una solución a la vez y no crea una imagen general del espacio de búsqueda.

Los algoritmos genéticos operan de forma simultánea con varias soluciones.

A2.2.5 Otras consideraciones

Estas son algunas otras consideraciones a hacer acerca de los algoritmos genéticos:

- **Resulta sumamente fácil ejecutarlos en las modernas arquitecturas masivas en paralelo.**
- **Usan operadores probabilistas, en vez de los deterministas clásicos.**
- **Pueden converger prematuramente debido a diversos tipos de problemas.**

A2.3 Conceptos Avanzados

Hasta el momento hemos presentado la estructura básica de un algoritmo genético. Estos pueden ser extendidos y tratados de varias formas mediante la utilización de ciertos conceptos avanzados.

A2.3.1 Representaciones no binarias

Tal como se mencionó anteriormente, la representación más general de un cromosoma es una cadena de bits. Sin embargo, se ha constatado que la codificación binaria puede no resultar adecuada por alguna de las siguientes razones **[B18]**:

- Epístasis: el valor de un bit puede suprimir las contribuciones de aptitud de otros bits en el genotipo.
- Representación natural: algunos problemas se prestan para la utilización de representaciones de mayor cardinalidad que la binaria (p.ej.: Problema del Viajero)
- Soluciones ilegales: los operadores genéticos utilizados pueden producir con frecuencia soluciones ilegales, necesitando de métodos para corregir estos errores.

Dentro de las posibles representaciones de un cromosoma (la cual dependerá estrictamente del problema y de los operadores que se utilicen sobre ella) se encuentran:

- Códigos de Gray: presenta un mapeo más adecuado del espacio de búsqueda con el de representación.
- Números reales: ha sido muy criticada por los teóricos de esta técnica, principalmente porque tiende a hacer que el comportamiento del algoritmo sea más errático y difícil de predecir. Sin embargo se ha demostrado empíricamente que el uso de esta representación funciona mejor que la binaria.
- Longitud variable: existen problemas en los cuales es conveniente el uso de un alfabeto de longitud variable para manejar los cambios que ocurran en el ambiente con respecto al tiempo. Con respecto a esto, se ha propuesto el uso de un algoritmo genético particular denominado desordenado (messy GA), de longitud variable con población de tamaño variable.
- Representación de árbol: es usado principalmente para representar un conjunto de instrucciones de un lenguaje de programación y la combinación de estos para generar nuevos programas (Programación Genética).
- Otras representaciones: existen otras tales como el uso de gramáticas, matrices o listas.

A2.3.2 Operadores avanzados

Hemos obviado varios fenómenos y operadores naturales interesantes con el motivo de introducir el algoritmo genético simple. La sobrecarga del algoritmo con otro tipo de operadores da resultados interesantes, dependiendo del problema, y principalmente, de la frecuencia con que estos operadores se apliquen.

Cromosomas Diploides y Dominancia

Los cromosomas diploides (diploidy) son aquellos que contienen dos pares de genes, en lugar de uno. Cromosomas diploides dan la ventaja a individuos en donde el ambiente puede cambiar en el tiempo. Teniendo dos genes se permite recordar dos "soluciones" diferentes. Uno de los dos genes es dominante , mientras que el otro es recesivo. Si las condiciones ambientales cambian, el dominante puede pasar a ser el otro. Este mecanismo es ideal si el ambiente alterna regularmente entre dos estados.

Operador de Nicho y especialización

Especialización (speciation) es el proceso en el cual una especie se diferencia en dos (o más) especies diferentes ocupando diferentes nichos (niches). Estos operadores están encaminados a mantener la diversidad de la población, de forma que cromosomas similares sustituyan solo a cromosomas similares, y son especialmente útiles en problemas con muchas soluciones (con este operador, se es capaz de hallar todos los máximos, dedicándose cada especie a un máximo).

Inversion y otros operadores de reordenamiento

El orden de los genes en un cromosoma es imprescindible para cumplir la hipótesis de los bloques de construcción (building blocks, es una de las hipótesis fundamentales de la teoría de los algoritmos genéticos que expresa la necesidad de mantener juntos aquellos genes que funcionen bien juntos). Existen técnicas para reordenar (reordering) la posición de los genes en el cromosoma durante la ejecución. Una de estas técnicas es la inversión (inversion), la cual trabaja invirtiendo el orden de los genes que se encuentran entre dos posiciones del cromosoma, elegidas aleatoriamente (generalmente, cuando esto se aplica, los genes son "etiquetados" con números para tenerlos correctamente identificados). El propósito del reordenamiento es el intento de encontrar genes con mayor potencial evolutivo. También expande el espacio de búsqueda.

Otros micro operadores

Existen otros operadores que permiten desde el control de la taza de mutación (duplicación y borrado), hasta la especialización y cooperación entre especies (determinación sexual y diferenciación).

A2.3.3 Procesamiento paralelo

Los algoritmos genéticos pueden ser paralelizados **[B19]** para ganar en performance y escalabilidad. Pueden ser fácilmente implementados en redes de computadoras heterogéneas o en mainframes paralelos.

La forma de paralelización de un algoritmo genético depende de los siguientes elementos:

- Cómo el fitness es evaluado y la mutación aplicada.
- Si son usadas poblaciones simples o múltiples subpoblaciones.
- Si se utilizan múltiples poblaciones, cómo los individuos son intercambiados.
- Cómo es aplicada la selección (global o local).

Dependiendo de cómo cada uno de estos elementos es implementado, se pueden obtener variados métodos de paralelización, los cuales pueden ser clasificados en las siguientes clases:

- Paralelización maestro-esclavo (evaluación de fitness distribuida), sincrónica o asincrónica.
- Subpoblaciones estáticas con migración (traspaso de individuos entre poblaciones).
- Subpoblaciones estáticas solapadas (sin migración, los individuos pueden pertenecer a más de una población).
- Algoritmos genéticos masivamente paralelos.
- Subpoblaciones dinámicas solapadas.
- Algoritmos genéticos paralelos de estado seguro (steady-state).
- Algoritmos genéticos paralelos desordenados (messy).

Métodos híbridos (e.j. subpoblaciones estáticas con migración y evaluación de fitness en cada población.).

Apéndice 3 Ejecución del Algoritmo

Los componentes del algoritmo se pueden dividir en tres grupos: la implementación de cada estructura, los datos provistos y el algoritmo en sí mismo. El primer y último de los componentes pueden ser considerados estáticos mientras que el segundo puede variarse para resolver varias instancias del problema. Para esto es necesario configurar los parámetros del algoritmo o generar los datos del problema en base a un conjunto de archivos, luego es necesario ejecutar el algoritmo en base a un conjunto de scripts.

A3.1 Formato de Archivos

A3.1.1 Configuración

El algoritmo es configurado a partir de un archivo de texto, el cual es indicado en la invocación de este y cargado por el componente "Configuration", tal como se ha explicado en el capítulo 3.

Esta configuración permite que el usuario defina cualquier tipo de información extra que desee ingresar al algoritmo siempre y cuando siga con determinada estructura:

NOMBRE_PARÁMETRO = VALOR_PARÁMETRO

Existen ciertos parámetros que el algoritmo necesita siempre entre los cuales se encuentran las tasas de cruzamiento , de mutación, tamaño de la población, etc, los cuales no es posible obviar, ni siquiera cambiar el nombre del parámetro.

Una vista de archivo de configuración sería la siguiente:

```
GLOBALDATA_FILE   = ./PATH1/globaldata.dat
RANDOM_FILE       = ./PATH2/nr.dat
FINALDATA_FILE    = ./PATH3/sal_pcmcf1_lc.dat
GENOTYPE_FILE     = ./PATH4/gen_pcmcf1_lc.dat

SORTEO                  =     0.8
GENERATIONS_NUMBER      =     100
GENOTYPE_LEN            =     20
POPULATION_SIZE         =     30
CROSSOVER_RATE          =     0.95
MUTATION_RATE           =     0.00333
SELECTION_RATE          =     0.7
FITNESS_TYPE            =     1  #0 - Diferencia,  1 - Inversa
CRITERIO_TYPE           =     1  #0 – Generaciones,  1 – Calidad
FACTOR_ESCALAMIENTO     =     2
PARAMETRONUEVO          =     VALOR
```

El símbolo # determina un comentario, los parámetros en negrita dependen del usuario mientras que el resto es estático.

A3.1.2 Representación del Problema

Como se ha explicado antes el problema y su solución se representan como un grafo. Con motivo de mantener cierta coherencia con **[B22]** se utilizó el mismo formato de archivo tanto para la lectura de los datos como para desplegarlos a la finalización de la ejecución.

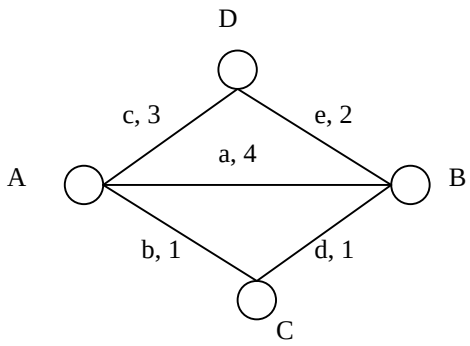
La formato es el siguiente:

```
-------------------------------------------------------------------------
vertice     <id_vertice>
vertice     <id_vertice>
.............
vertice     <id_vertice>
arista      <id_ari>    <id_vertice>      <id_vertice>      <costo>
.............
arista      <id_ari>    <id_vertice>      <id_vertice>      <costo>
arista      <id_ari>    <id_vertice>      <id_vertice>      <costo>
restriccion <número>    <id_vertice>      <id_vertice>
restriccion <número>    <id_vertice>      <id_vertice>
.............
restriccion <número>    <id_vertice>      <id_vertice>
-------------------------------------------------------------------------
```

<id_vertice>, <id_ari> son strings que identifican los vértices y aristas del grafo.
<número> son enteros y representan las restricciones que se imponen entre el par de vértices terminales.

Ejemplo:

```
-----------------------------------------------------
vertice     A
vertice     B
vertice     C
vertice     D
arista      a      A      B      4
arista      b      A      C      1
arista      c      A      D      3
arista      d      B      C      1
arista      e      B      D      2
restriccion 1      A      D
restriccion 2      A      B
-----------------------------------------------------
```



A3.1.3 Salida

La salida se almacenará en los archivos indicados en el archivo de configuración. El formato de los datos de salida será el siguiente.

A3.1.3.1 Datos de Salida

Además de la representación del grafo final con el formato indicado anteriormente, se almacenan los datos estadísticos con el siguiente formato:

```
Bytes promedio de individuo: NUM
Bytes de la población: NUM
Velocidad de creacion de individuo promedio (ms): NUM
Velocidad de creacion de poblacion (ms): NUM

Tiempo promedio de cruza (ms): NUM
Tiempo total de cruza (ms): NUM
Tiempo promedio de mutacion (ms): NUM
Tiempo total de mutacion (ms): NUM

Tiempo de ejecucion (ms): NUM
Tiempo de evolucion (ms): NUM
Cantidad de Generaciones: NUM
Fitness del mejor genotipo: NUM
Iteracion del mejor genotipo: NUM

(Tiempo,Iteracion,Mejor Fitness,Fitness Promedio)
Tiempo_1    Iter_1    Mej_Fit_1    Fit_Prom_1
Tiempo_2    Iter_2    Mej_Fit_2    Fit_Prom_2
.........................................
Tiempo_N    Iter_N    Mej_Fit_N    Fit_Prom_N
```

A3.2 Ejecución de las Pruebas

El ejecutable es creado a través del siguiente Makefile, el cual permite además comprimir los datos de las pruebas y sus resultados, comprimir los fuentes de la aplicación y borrar los ejecutables.

```
OPCIONES = -Wall -g -lm
CXXFLAGS = $(OPCIONES)
C = gcc
CXX = $(C)
LINKER = $(OPCIONES)

FUENTES = Configuracion/*.c Configuracion/*.h Grafo/*.c Grafo/*.h
          Population/*.c Population/*.h $(REPB) $(REPLC) $(REPSR)
          Utils/*.c Utils/*.h

FUENC = Configuracion/*.c Grafo/*.c Population/*.c Utils/*.c
PRUEBAS = Pruebas/*
REPB = Rep_Binaria/genotype.c Rep_Binaria/genotype.h
REPLC = Rep_LisCam/genotype.c Rep_LisCam/genotype.h
REPSR = Rep_SubReq/genotype.c Rep_SubReq/genotype.h
INCLUDE = -IConfiguracion -IGrafo -IPopulation -IUtils
VALIDACION = valid_binaria valid_liscam valid_subreq
all:
      make motorBinario
      make motorLisCam
      make motorSubReq

motorBinario: $(FUENTES) $(REPB) motor.c
      $(C) $(LINKER) $(FUENC) Rep_Binaria/genotype.c motor.c -o
                              motorBinario $(INCLUDE) -IRep_Binaria

motorLisCam: $(FUENTES) $(REPLC) motor.c
      $(C) $(LINKER) $(FUENC) Rep_LisCam/genotype.c motor.c -o
                              motorLisCam $(INCLUDE) -IRep_LisCam

motorSubReq: $(FUENTES) $(REPSR) motor.c
      $(C) $(LINKER) $(FUENC) Rep_SubReq/genotype.c motor.c -o
                              motorSubReq $(INCLUDE) -IRep_SubReq
rebuild:
      make clean
      make
clean:
      -rm -f *~ core motorBinario motorLisCam motorSubReq
      make zip
      make pruebas
zip:
      rm -f codigo.tar.gz
      tar -cvf codigo.tar $(FUENTES) makefile motor.c $(VALIDACION)
      gzip codigo.tar

pruebas:
      rm -f pruebas.tar.gz
      tar -cvf pruebas.tar $(PRUEBAS)
      gzip pruebas.tar
```

A3.3 Resultado de las Pruebas

Estos son los resultados obtenidos de las diferentes corridas del algoritmo genético necesarias para las pruebas (validación, configuración y comparación).

A3.3.1 Validación

La representación gráfica de los resultados se puede observar en la siguiente tabla:

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(Fin?,Costo,Tiempo,#Iter,#IterBest)	.....
	Diferencia	.....	.....

En ella se indica si el algoritmo temina o nó (1 - sí, 0 - no), el costo de la solución obtenida, el tiempo de ejecución (ms), cantidad de iteraciones e iteración en la cual se encontró la mejor solución.

A3.3.1.1 Binaria

Problema1 (PV1)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,5,73,27,13)	(1,5,226,100,13)
	Diferencia	(1,5,54,21,7)	(1,5,229,100,7)

Problema 2 (PV2)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,7,60,15,1)	(1,7,371,100,1)
	Diferencia	(1,7,62,15,1)	(1,7,383,100,1)

Problema 3 (PV3)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,9,283,17,3)	(1,9,1160,100,10)
	Diferencia	(1,9,176,16,2)	(1,9,1392,100,2)

Problema 4 (PV4)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,20,144,24,10)	(1,21,610,100,10)
	Diferencia	(1,20,128,21,7)	(1,21,607,100,7)

Problema 5 (PV5)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,24,290,22,8)	(1,24,1219,100,8)
	Diferencia	(1,24,221,15,1)	(1,24,1188,100,1)

Problema 6 (PV6)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,40,808,15,1)	(1,40,5161,100,1)
	Diferencia	(1,39,1167,21,7)	(1,39,5148,100,7)

Problema 7 (PV7)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,11,73,17,3)	(1,9,375,100,56)
	Diferencia	(1,11,101,27,13)	(1,11,354,100,13)

Problema 8 (PV8)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,143,50533,13,9)	(1,115,1748239,500,327)
	Diferencia	(1,100,397505,115,101)	(1,100,1640744,500,116)

Problema 9 (PV9)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,2889,272930,100,86)	(1,2012,1308075,500,490)
	Diferencia	(1,1499,328820,124,110)	(1,646,1201924,500,425)

A3.3.1.2 Lista de Caminos

Problema1 (PV1)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,5,75,13,1)	(1,5,473,100,1)
	Diferencia	(1,5,67,13,1)	(1,5,457,100,1)

Problema 2 (PV2)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,8,121,13,1)	(1,8,784,100,1)
	Diferencia	(1,8,128,13,1)	(1,8,792,100,1)

Problema 3 (PV3)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,10,466,13,1)	(1,10,3346,100,1)
	Diferencia	(1,10,471,13,1)	(1,10,3435,100,1)

Problema 4 (PV4)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,22,235,13,1)	(1,22,1612,100,1)
	Diferencia	(1,22,240,13,1)	(1,22,1588,100,1)

Problema 5 (PV5)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,27,631,13,1)	(1,27,4374,100,1)
	Diferencia	(1,27,661,13,1)	(1,27,4358,100,1)

Problema 6 (PV6)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,43,1908,13,1)	(1,43,13609,100,1)
	Diferencia	(1,39,2302,16,2)	(1,39,13710,100,2)

Problema 7 (PV7)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,9,109,13,1)	(1,9,729,100,1)
	Diferencia	(1,13,120,13,1)	(1,13,777,100,1)

Problema 8 (PV8)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,127,90823,13,1)	(1,127,7296530,1000,1)
	Diferencia	(1,122,91606,13,1)	(1,122,6925448,1000,1)

Problema 9 (PV9)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,632,24755,13,1)	(1,615,2037474,1000,522)
	Diferencia	(1,551,173411,93,79)	(1,470,2042254,1000,939)

A3.3.1.3 Subgrafo de Requerimientos

Problema1 (PV1)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,5,75,13,1)	(1,5,527,100,1)
	Diferencia	(1,5,77,13,1)	(1,5,533,100,1)

Problema 2 (PV2)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,8,138,13,1)	(1,8,1003,100,1)
	Diferencia	(1,8,148,13,1)	(1,8,968,100,1)

Problema 3 (PV3)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,10,646,13,1)	(1,10,5104,100,1)
	Diferencia	(1,10,673,13,1)	(1,10,5389,100,1)

Problema 4 (PV4)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,22,276,13,1)	(1,22,2070,100,1)
	Diferencia	(1,22,298,13,1)	(1,22,2079,100,1)

Problema 5 (PV5)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,25,699,13,1)	(1,25,5121,100,1)
	Diferencia	(1,25,710,13,1)	(1,25,710,13,1)

Problema 6 (PV6)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,43,2212,13,1)	(1,43,16759,100,1)
	Diferencia	(1,39,2703,16,2)	(1,39,16622,100,2)

Problema 7 (PV7)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,9,149,15,1)	(1,9,908,100,1)
	Diferencia	(1,9,148,15,1)	(1,9,887,100,1)

Problema 8 (PV8)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,127,98530,13,1)	(1,127,7520347,1000,1)
	Diferencia	(1,122,98788,13,1)	(1,122,7597067,1000,1)

Problema 9 (PV9)

		Criterio de Parada	
		Calidad de Solución	Esfuerzo Predefinido
Fitness	Inverso	(1,632,29766,13,1)	(1,615,2466717,1000,522)
	Diferencia	(1,551,213358,93,79)	(1,470,2462300,1000,939)

A3.3.2 Configuración

A3.3.2.1 Cálculo de Fitness

Cada tupla indica: costo de la solución, tiempo de ejecución (ms), cantidad de iteraciones e iteración en la cual se encontró la mejor solución.

		PCf1	PCf2	PCf3	PCf4	PCf5
Fitness	Inverso	(43,18152,100,16)	(25,736,100,100)	(56,9255,100,10)	(101,869231,500,475)	(1994,1324173,500,276)
	Diferencia	(43,17066,100,13)	(20,643,100,41)	(45,9041,100,84)	(88,816965,500,127)	(1470,1303211,500,147)

A3.3.2.2 Tamaño de la Población

Cada tupla de la tabla indica: costo de la solución, tiempo de ejecución (ms), cantidad de iteraciones e iteración en la cual se encontró la mejor solución.

	PCf5
60	(509,1468800,500,264)
120	(498,3048049,500,382)
180	(492,4729891,500,389)

A3.3.2.3 Tasas de Cruza y Mutación

Cada tupla de la tabla indica: costo de la solución, tiempo de ejecución (ms),cantidad de iteraciones e iteración en la cual se encontró la mejor solución. Los valores a y b corresponden a diferentes secuencias de números aleatorios.

Binaria

	Cruza			
	PCf4 (a)	PCf5 (a)	PCf4 (b)	PCf5 (b)
0.75	(84,455338,300,187)	(1473,745964,300,298)	(99,442899,300,234)	(955,711201,300,294)
0.8	(82,464495,300,267)	(1541,759422,300,179)	(101,449606,300,292)	(928,725280,300,300)
0.85	(85,476690,300,115)	(1506,774663,300,129)	(98,485483,300,250)	(971,741942,300,297)
0.9	(88,493434,300,127)	(1470,790788,300,147)	(102,475580,300,145)	(914,757366,300,300)

Mutación

	PCf4 (a)	PCf5 (a)	PCf4 (b)	PCf5 (b)
0.001	(84,458276,300, 219)	(1053,772322, 300,296)	(94,484857,300, 106)	(1145,775012, 300,299)
0.003	(88,493305,300, 127)	(956,752602, 300,293)	(102,475531,300, 145)	(1339,777238, 300,271)
0.006	(80,490085,300, 132)	(1109,759725, 300,299)	(89,527542,300, 140)	(930,812726, 300,300)
0.01	(86,517129,300, 171)	(1559,792670, 300,216)	(83,521409,300, 205)	(1446,783615, 300,239)

Lista de Caminos

Cruza

	PCf4 (a)	PCf5 (a)	PCf4 (b)	PCf5 (b)
0.75	(122,645491, 300,1)	(517,560585, 300,200)	(127,587508, 300,1)	(592,545333, 300,278)
0.8	(122,642742, 300,1)	(539,562496, 300,220)	(127,590675, 300,1)	(574,545169, 300,298)
0.85	(122,645135, 300,1)	(513,563494, 300,220)	(127,592346, 300,1)	(549,545691, 300,278)
0.9	(122,639069, 300,1)	(525,558358, 300,298)	(127,595729, 300,1)	(568,553842, 300,278)

Mutación

	PCf4 (a)	PCf5 (a)	PCf4 (b)	PCf5 (b)
0.001	(122,642404, 300,1)	(525,561416, 300,298)	(127,586095, 300,1)	(652,545826, 300,196)
0.003	(122,1278584, 300,1)	(525,557749, 300,298)	(127,1182228, 300,1)	(568,557989, 300,278)
0.006	(122,663311, 300,1)	(506,566804, 300,292)	(127,609369, 300,1)	(558,550601, 300,288)
0.01	(122,694057, 300,1)	(506,566154, 300,292)	(127,626299, 300,1)	(542,555287, 300,279)

Subgrafo de Requerimientos

Cruza

	PCf4 (a)	PCf5 (a)	PCf4 (b)	PCf5 (b)
0.75	(122,778114, 300,1)	(517,696737, 300,200)	(127,776707, 300,1)	(592,676793, 300,278)
0.8	(122,776821, 300,1)	(539,696833, 300,220)	(127,778754, 300,1)	(574,676171, 300,298)
0.85	(122,780962, 300,1)	(513,700296, 300,220)	(127,785968, 300,1)	(549,681286, 300,278)
0.9	(122,785100, 300,1)	(525,702792, 300,298)	(127,788039, 300,1)	(568,685196, 300,278)

Mutación

	PCf4 (a)	PCf5 (a)	PCf4 (b)	PCf5 (b)
0.001	(122,773947, 300,1)	(525,703071, 300,298)	(127,777771, 300,1)	(652,684071, 300,196)
0.003	(122,1566470, 300,1)	(525,702549, 300,298)	(127,1572995, 300,1)	(568,687256, 300,278)
0.006	(122,832983, 300,1)	(506,713734, 300,292)	(127,802628, 300,1)	(558,687064, 300,288)
0.01	(122,838562, 300,1)	(506,716629, 300,292)	(127,804615, 300,1)	(542,691883, 300,279)

A3.3.2.4 Criterio de Parada

Cada tupla indica: costo de la solución, tiempo de ejecución (ms), cantidad de iteraciones e iteración en la cual se encontró la mejor solución.

Binaria

		PCf1	PCf2	PCf3	PCf4	PCf5
Criterio	Esfuerzo Pred.	(43,16550,100,13)	(20,638,100, 41)	(45,8962,100, 84)	(88,810031, 500,127)	(1470,1300821, 500,147)
	Calidad	(43,4440,27, 13)	(20,367,55,41)	(51,2361,26, 12)	(89,212169, 126,112)	(1669,224687, 84,70)

Lista de Caminos

		PCf1	PCf2	PCf3	PCf4	PCf5
Criterio	Esfuerzo Pred.	(50,30390,100,14)	(19,1237,100, 1)	(69,7405,100, 1)	(122,1073769, 500,1)	(511,801041, 500,25)
	Calidad	(50,8545,13,6)	(19,198,13,1)	(69,2064,13,1)	(122,60464,13,1)	(551,343332, 93,79)

Subgrafo de Requerimientos

		PCf1	PCf2	PCf3	PCf4	PCf5
Criterio	Esfuerzo Pred.	(46,47494,100,1)	(19,1395,100, 1)	(62,9984,100, 100)	(122,1326801,500,1)	(511,1211963, 500,458)
	Calidad	(46,11974,13, 1)	(19,210,13,1)	(69,2494,13,1)	(122,69917,13,1)	(551,423986, 93,79)

A3.3.3 Comparación

Los tiempos de las tablas están en milisegundos, los carácteres de la A a la E se refieren a diferentes secuencias de números aleatorios. La media y desviación estándar del tiempo está en segundos.

A3.3.3.1 Problema 1 (PCm1)

Binaria	A	B	C	D	E
Bytes promedio de individuo	19812	19812	19812	19812	19812
Bytes total de población	2377440	2377440	2377440	2377440	2377440
Tiempo promedio de creación	37.25	31.675	31.4083	31.525	31.5333
Tiempo total de creación	4470	3801	3769	3783	3784
Tiempo de cruza promedio	2451.11	2440.389	2533.615	2411.721	2455.003
Tiempo de cruza total	737784	734557	762618	725928	738956
Tiempo de mutación promedio	4150.937	4234.369	4482.691	4131.06	4216.462
Tiempo de mutación total	1249432	1274545	1349290	1243449	1269155
Tiempo total de ejecución	1994547	2015822	2118493	1976005	2015146
Tiempo total de evolución	1990019	2011981	2114684	1972181	2011322
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(941,225)	(1119,265)	(1481,293)	(903,293)	(1187,291)

Binaria	F	G	H	I	J
Bytes promedio de individuo	19812	19812	19812	19812	19812
Bytes total de población	2377440	2377440	2377440	2377440	2377440
Tiempo promedio de creación	31.3917	31.8333	31.3833	31.4833	31.6167
Tiempo total de creación	3767	3820	3766	3778	3794
Tiempo de cruza promedio	2480.299	2400.897	2432.728	2460.216	2382.568
Tiempo de cruza total	746570	722670	732251	740525	717153
Tiempo de mutación promedio	4615.748	4113.435	4192.468	4095.635	4174.595
Tiempo de mutación total	1389340	1238144	1261933	1232786	1256553
Tiempo total de ejecución	2142792	1968106	2000972	1980134	1980546
Tiempo total de evolución	2138961	1964221	1997152	1976294	1976698
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(1561,265)	(952,291)	(1016,261)	(902,273)	(946,278)

	Media	Desviación Estándar
Tiempo (seg)	2020	61
Costo	1100.8	240.73

Lista de Caminos	A	B	C	D	E
Bytes promedio de individuo	1824.4	1816.8	1791.133	1819.333	1823.8
Bytes total de población	218928	218016	214936	218320	218856
Tiempo promedio de creación	55.8	55.7417	56.65	56.0583	55.1667
Tiempo total de creación	6696	6689	6798	6727	6620
Tiempo de cruza promedio	26.7542	27.5282	27.5748	26.9435	27.8372
Tiempo de cruza total	8053	8286	8300	8110	8379
Tiempo de mutación promedio	4691.482	4642.957	4906.09	4616.871	4674.904
Tiempo de mutación total	1412136	1397530	1476733	1389678	1407146
Tiempo total de ejecución	1430974	1416568	1495836	1408582	1426366
Tiempo total de evolución	1424241	1409842	1489001	1401817	1419707
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(537,296)	(529,295)	(546,251)	(553,103)	(567,247)

Lista de Caminos	F	G	H	I	J
Bytes promedio de individuo	1820.933	1824.2	1817.267	1825.667	1822.333
Bytes total de población	218512	218904	218072	219080	218680
Tiempo promedio de creación	62.275	62.5333	62.7167	63.1833	63.7417
Tiempo total de creación	7473	7504	7526	7582	7649
Tiempo de cruza promedio	31.1395	29.196	29.3056	29.2425	29.9601
Tiempo de cruza total	9373	8788	8821	8802	9018
Tiempo de mutación promedio	5508.535	5296.335	5327.748	5305.246	5270.236
Tiempo de mutación total	1658069	1594197	1603652	1596879	1586341
Tiempo total de ejecución	1679310	1615278	1624358	1617513	1607502
Tiempo total de evolución	1671777	1607299	1616791	1609879	1599811
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(514,233)	(542,298)	(546,242)	(623,1)	(568,293)

	Media	Desviación Estándar
Tiempo (seg)	1532	106
Costo	552.5	29.61

Subgrafo de Requerimientos	A	B	C	D	E
Bytes promedio de individuo	660	660	660	660	660
Bytes total de población	79200	79200	79200	79200	79200
Tiempo promedio de creación	71.1833	72.3833	73.0417	72.0917	70.2417
Tiempo total de creación	8542	8686	8765	8651	8429
Tiempo de cruza promedio	345.9535	344.2027	360.5183	342.6711	342.6412
Tiempo de cruza total	104132	103605	108516	103144	103135
Tiempo de mutación promedio	4016.096	4038.159	4232.708	4005.947	4062.007
Tiempo de mutación total	1208845	1215486	1274045	1205790	1222664
Tiempo total de ejecución	1336170	1342485	1406308	1332253	1349115
Tiempo total de evolución	1327592	1333761	1397504	1323565	1340648
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(537,296)	(529,295)	(546,251)	(553,103)	(567,247)

Subgrafo de Requerimientos	F	G	H	I	J
Bytes promedio de individuo	660	660	660	660	660
Bytes total de población	79200	79200	79200	79200	79200
Tiempo promedio de creación	81.65	80.6083	83.0917	82.0417	82.1833
Tiempo total de creación	9798	9673	9971	9845	9862
Tiempo de cruza promedio	405.309	388.505	392.3422	396.7176	391.2558
Tiempo de cruza total	121998	116940	118095	119412	117768
Tiempo de mutación promedio	5144.053	4605.385	4657.083	4558.837	4631.851
Tiempo de mutación total	1548360	1386221	1401782	1372210	1394187
Tiempo total de ejecución	1697328	1528812	1545955	1517291	1538030
Tiempo total de evolución	1687480	1519088	1535933	1507394	1528116
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(514,233)	(542,298)	(546,242)	(623,1)	(568,293)

	Media	Desviación Estándar
Tiempo (seg)	1459	124
Costo	552.5	29.61

A3.3.3.2 Problema 2 (PCm2)

Binaria	A	B	C	D	E
Bytes promedio de individuo	19812	19812	19812	19812	19812
Bytes total de población	2377440	2377440	2377440	2377440	2377440
Tiempo promedio de creación	40.1083	40.4	40.7333	40.5583	40.375
Tiempo total de creación	4813	4848	4888	4867	4845
Tiempo de cruza promedio	3143.515	3192.508	3364.698	3188.973	3327.588
Tiempo de cruza total	946198	960945	1012774	959881	1001604
Tiempo de mutación promedio	5119.538	5181.236	5399.638	5129.212	5347.718
Tiempo de mutación total	1540981	1559552	1625291	1543893	1609663
Tiempo total de ejecución	2494859	2528205	2645808	2511517	2619076
Tiempo total de evolución	2489993	2523308	2640871	2506569	2614165
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(904,278)	(909,220)	(1353,279)	(870,265)	(1096,300)

Binaria	F	G	H	I	J
Bytes promedio de individuo	19812	19812	19812	19812	19812
Bytes total de población	2377440	2377440	2377440	2377440	2377440
Tiempo promedio de creación	40.1667	40.5333	40.2667	40.025	40.4583
Tiempo total de creación	4820	4864	4832	4803	4855
Tiempo de cruza promedio	3288.648	3178.027	3141.581	3223.362	3147.585
Tiempo de cruza total	989883	956586	945616	970232	947423
Tiempo de mutación promedio	5497.768	5098.635	5112.226	5088.236	5148.246
Tiempo de mutación total	1654828	1534689	1538780	1531559	1549622
Tiempo total de ejecución	2652531	2499169	2492266	2509622	2504914
Tiempo total de evolución	2647661	2494255	2487371	2504769	2500008
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(1451,273)	(850,277)	(899,224)	(792,275)	(979,298)

	Media	Desviación Estándar
Tiempo (seg)	2546	66
Costo	1010.3	222.85

Lista de Caminos	A	B	C	D	E
Bytes promedio de individuo	2271.067	2272.467	2231	2271.733	2269.267
Bytes total de población	272528	272696	267720	272608	272312
Tiempo promedio de creación	74.3417	72.3333	74.7917	73.4583	73.8
Tiempo total de creación	8921	8680	8975	8815	8856
Tiempo de cruza promedio	33.6678	33.6246	33.2126	32.3522	34
Tiempo de cruza total	10134	10121	9997	9738	10234
Tiempo de mutación promedio	5920.983	6031.203	6480.661	5976.532	6071.88
Tiempo de mutación total	1782216	1815392	1950679	1798936	1827636
Tiempo total de ejecución	1806511	1839389	1974657	1822537	1852022
Tiempo total de evolución	1797539	1830664	1965637	1813678	1843120
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(647,107)	(646,244)	(652,297)	(660,282)	(651,277)

Lista de Caminos	F	G	H	I	J
Bytes promedio de individuo	2270.533	2276.733	2272	2271.667	2269.8
Bytes total de población	272464	273208	272640	272600	272376
Tiempo promedio de creación	84.3167	82.6583	88.3083	82.3083	83.6583
Tiempo total de creación	10118	9919	10597	9877	10039
Tiempo de cruza promedio	36.4917	36.3023	36.6844	36.495	36.6877
Tiempo de cruza total	10984	10927	11042	10985	11043
Tiempo de mutación promedio	7321.645	6678.103	6796.857	6710.485	6904.804
Tiempo de mutación total	2203815	2010109	2045854	2019856	2078346
Tiempo total de ejecución	2230403	2036481	2073029	2046724	2104893
Tiempo total de evolución	2220222	2026501	2062382	2036795	2094804
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(609,286)	(682,228)	(607,271)	(669,174)	(652,298)

	Media	Desviación Estándar
Tiempo (seg)	1979	144
Costo	647.5	23.54

Subgrafo de Requerimientos	A	B	C	D	E
Bytes promedio de individuo	876	876	876	876	876
Bytes total de población	105120	105120	105120	105120	105120
Tiempo promedio de creación	90.225	93.3083	97.5417	93.0583	94.65
Tiempo total de creación	10827	11197	11705	11167	11358
Tiempo de cruza promedio	450.4551	454.2259	471.598	449.8837	454.1462
Tiempo de cruza total	135587	136722	141951	135415	136698
Tiempo de mutación promedio	5111.452	5210.578	5579.06	5123.628	5165.362
Tiempo de mutación total	1538547	1568384	1679297	1542212	1554774
Tiempo total de ejecución	1703921	1735378	1852409	1707646	1721830
Tiempo total de evolución	1693048	1724134	1840659	1696432	1710425
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(663,276)	(620,299)	(665,287)	(686,251)	(669,204)

Subgrafo de Requerimientos	F	G	H	I	J
Bytes promedio de individuo	876	876	876	876	876
Bytes total de población	105120	105120	105120	105120	105120
Tiempo promedio de creación	109.9333	110.15	104.075	106.6167	108.9333
Tiempo total de creación	13192	13218	12489	12794	13072
Tiempo de cruza promedio	526.093	510.7542	514.2592	511.6844	512.7641
Tiempo de cruza total	158354	153737	154792	154017	154342
Tiempo de mutación promedio	6562.515	5946.246	5915.392	5943.701	5925.714
Tiempo de mutación total	1975317	1789820	1780533	1789054	1783640
Tiempo total de ejecución	2168692	1977314	1968352	1976962	1971707
Tiempo total de evolución	2155432	1964033	1955812	1964118	1958578
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(617,180)	(680,167)	(671,127)	(664,217)	(655,298)

	Media	Desviación Estándar
Tiempo (seg)	1878	158
Costo	659	23.07

A3.3.3.3 Problema 3 (PCm3)

Binaria	A	B	C	D	E
Bytes promedio de individuo	19812	19812	19812	19812	19812
Bytes total de población	2377440	2377440	2377440	2377440	2377440
Tiempo promedio de creación	38.7917	38.8	38.1583	38.6833	39.3083
Tiempo total de creación	4655	4656	4579	4642	4717
Tiempo de cruza promedio	2772.648	2775.462	3057.422	2808.983	2799.721
Tiempo de cruza total	834567	835414	920284	845504	842716
Tiempo de mutación promedio	4630.757	4684.555	4908.847	4625.179	4711.269
Tiempo de mutación total	1393858	1410051	1477563	1392179	1418092
Tiempo total de ejecución	2236003	2253042	2405329	2245251	2268440
Tiempo total de evolución	2231301	2248340	2400704	2240563	2263677
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(939,300)	(1089,296)	(1422,260)	(966,286)	(1230,288)

Binaria	F	G	H	I	J
Bytes promedio de individuo	19812	19812	19812	19812	19812
Bytes total de población	2377440	2377440	2377440	2377440	2377440
Tiempo promedio de creación	45.075	44.1833	44.6583	44.3833	47.575
Tiempo total de creación	5409	5302	5359	5326	5709
Tiempo de cruza promedio	3365.934	3364.346	3280.844	3472.681	3375.455
Tiempo de cruza total	1013146	1012668	987534	1045277	1016012
Tiempo de mutación promedio	5773.409	5274.495	5298.299	5258.472	5282.704
Tiempo de mutación total	1737796	1587623	1594788	1582800	1590094
Tiempo total de ejecución	2759356	2608642	2590719	2636427	2614860
Tiempo total de evolución	2753896	2603288	2585307	2631049	2609099
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(1575,236)	(907,293)	(981,284)	(959,261)	(918,263)

	Media	Desviación Estándar
Tiempo (seg)	2462	201
Costo	1098.6	234.43

Lista de Caminos	A	B	C	D	E
Bytes promedio de individuo	2332	2327.733	2292	2326.067	2327.667
Bytes total de población	279840	279328	275040	279128	279320
Tiempo promedio de creación	80.4583	81.9583	79.7333	82.0583	80.375
Tiempo total de creación	9655	9835	9568	9847	9645
Tiempo de cruza promedio	34.1395	33.3455	32.7409	33.6445	35.299
Tiempo de cruza total	10276	10037	9855	10127	10625
Tiempo de mutación promedio	6501.379	6551.877	6781.249	6511.183	6503.582
Tiempo de mutación total	1956915	1972115	2041156	1959866	1957578
Tiempo total de ejecución	1982738	1997643	2065862	1985569	1983897
Tiempo total de evolución	1973037	1987764	2056245	1975674	1974205
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(619,228)	(582,300)	(598,297)	(639,205)	(650,42)

Lista de Caminos	F	G	H	I	J
Bytes promedio de individuo	2325.867	2333.667	2322.467	2325.667	2325.267
Bytes total de población	279104	280040	278696	279080	279032
Tiempo promedio de creación	94.225	90.175	94.1333	91.1167	91.275
Tiempo total de creación	11307	10821	11296	10934	10953
Tiempo de cruza promedio	35.1528	36.3355	36.309	37.2591	34.8239
Tiempo de cruza total	10581	10937	10929	11215	10482
Tiempo de mutación promedio	7735.774	7467.628	7481.113	7465.983	7451.455
Tiempo de mutación total	2328468	2247756	2251815	2247261	2242888
Tiempo total de ejecución	2356078	2275646	2280082	2275590	2270205
Tiempo total de evolución	2344717	2264772	2268733	2264603	2259201
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(603,282)	(573,293)	(625,248)	(627,109)	(568,300)

	Media	Desviación Estándar
Tiempo (seg)	2147	156
Costo	608.4	28.13

Subgrafo de Requerimientos	A	B	C	D	E
Bytes promedio de individuo	912	912	912	912	912
Bytes total de población	109440	109440	109440	109440	109440
Tiempo promedio de creación	105.4667	102.3833	101.3667	97.2167	103.9583
Tiempo total de creación	12656	12286	12164	11666	12475
Tiempo de cruza promedio	469.7774	471.1163	494.8206	471.907	469.1761
Tiempo de cruza total	141403	141806	148941	142044	141222
Tiempo de mutación promedio	6342.309	7860.907	6668.256	6266.136	6391.691
Tiempo de mutación total	1909035	2366133	2007145	1886107	1923899
Tiempo total de ejecución	2082869	2540213	2189168	2059610	2097653
Tiempo total de evolución	2070168	2527881	2176958	2047898	2085131
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(561,267)	(575,209)	(597,201)	(586,211)	(649,119)

Subgrafo de Requerimientos	F	G	H	I	J
Bytes promedio de individuo	912	912	912	912	912
Bytes total de población	109440	109440	109440	109440	109440
Tiempo promedio de creación	117.2917	114.9833	116.4083	120.4833	121.1583
Tiempo total de creación	14075	13798	13969	14458	14539
Tiempo de cruza promedio	553.9136	533.103	538.6047	536.8372	541.9568
Tiempo de cruza total	166728	160464	162120	161588	163129
Tiempo de mutación promedio	8293.239	7181.023	7333.797	7194.94	7279.442
Tiempo de mutación total	2496265	2161488	2207473	2165677	2191112
Tiempo total de ejecución	2700505	2357062	2404999	2363202	2390174
Tiempo total de evolución	2686379	2343210	2390977	2348693	2375582
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(567,280)	(599,293)	(562,294)	(604,167)	(570,269)

	Media	Desviación Estándar
Tiempo (seg)	2318	211
Costo	587	26.9

A3.3.3.4 Problema de Configuracion 4 (PCf4)

Binaria	A	B	C	D	E
Bytes promedio de individuo	8076	8076	8076	8076	8076
Bytes total de población	969120	969120	969120	969120	969120
Tiempo promedio de creación	431.6417	708.6917	134.1333	594.4167	565.6334
Tiempo total de creación	51797	85043	16096	71330	67876
Tiempo de cruza promedio	1467.887	1482.273	1355.432	1427.169	1449.595
Tiempo de cruza total	441834	446164	407985	429578	436328
Tiempo de mutación promedio	2334.445	2390.12	2680.236	2343.312	2425.508
Tiempo de mutación total	702668	719426	806751	705337	730078
Tiempo total de ejecución	1197580	1251908	1232113	1207525	1235586
Tiempo total de evolución	1145756	1166839	1215992	1136170	1167685
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(83,143)	(83,164)	(83,300)	(84,281)	(83,210)

Binaria	F	G	H	I	J
Bytes promedio de individuo	8076	8076	8076	8076	8076
Bytes total de población	969120	969120	969120	969120	969120
Tiempo promedio de creación	656.5917	417.925	586.3333	628.9417	585.575
Tiempo total de creación	78791	50151	70360	75473	70269
Tiempo de cruza promedio	1255.239	1488.827	1494.864	1540.668	1440.522
Tiempo de cruza total	377827	448137	449954	463741	433597
Tiempo de mutación promedio	2719.711	2361.88	2344.11	2328.538	2351.386
Tiempo de mutación total	818633	710926	705577	700890	707767
Tiempo total de ejecución	1276516	1210495	1227212	1241392	1212913
Tiempo total de evolución	1197697	1160318	1156826	1165893	1142619
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(88,172)	(86,261)	(84,130)	(85,126)	(84,172)

	Media	Desviación Estándar
Tiempo (seg)	1229	24
Costo	84.3	1.64

Lista de Caminos	A	B	C	D	E
Bytes promedio de individuo	7640.4	7787.467	7371.6	7676.8	7629.6
Bytes total de población	916848	934496	884592	921216	915552
Tiempo promedio de creación	60.275	124.225	61.5917	78.3167	76.9167
Tiempo total de creación	7233	14907	7391	9398	9230
Tiempo de cruza promedio	128.9335	190.196	82.8837	88.1993	88.4352
Tiempo de cruza total	38809	57249	24948	26548	26619
Tiempo de mutación promedio	5245.784	7417.582	4135.249	5993.013	5954.694
Tiempo de mutación total	1578981	2232692	1244710	1803897	1792363
Tiempo total de ejecución	1651893	2345141	1293140	1857671	1846069
Tiempo total de evolución	1644619	2330201	1285713	1848225	1836802
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(110,1)	(113,1)	(126,182)	(111,1)	(112,1)

Lista de Caminos	F	G	H	I	J
Bytes promedio de individuo	7761.067	7607.467	7673.6	7637.733	7635.867
Bytes total de población	931328	912896	920832	916528	916304
Tiempo promedio de creación	61.25	57.625	60.9417	61.65	56.8917
Tiempo total de creación	7350	6915	7313	7398	6827
Tiempo de cruza promedio	61.691	60.4552	61.7907	64.0897	60.7076
Tiempo de cruza total	18569	18197	18599	19291	18273
Tiempo de mutación promedio	5079.356	4814.412	5057.409	4720.542	4998.432
Tiempo de mutación total	1528886	1449138	1522280	1420883	1504528
Tiempo total de ejecución	1566126	1485649	1559863	1460057	1540962
Tiempo total de evolución	1558744	1478706	1552524	1452630	1534107
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(115,1)	(121,1)	(119,1)	(113,1)	(113,1)

	Media	Desviación Estándar
Tiempo (seg)	1661	295
Costo	115.3	5.10

Subgrafo de Requerimientos	A	B	C	D	E
Bytes promedio de individuo	1308	1308	1308	1308	1308
Bytes total de población	156960	156960	156960	156960	156960
Tiempo promedio de creación	63.0083	69.4583	65.9417	82.5333	80.9667
Tiempo total de creación	7561	8335	7913	9904	9716
Tiempo de cruza promedio	636.8771	722.608	572.7841	563.5415	556.8073
Tiempo de cruza total	191700	217505	172408	169626	167599
Tiempo de mutación promedio	5700.545	5944.066	4372.797	6121.179	6090.628
Tiempo de mutación total	1715864	1789164	1316212	1842475	1833279
Tiempo total de ejecución	1987206	2091741	1545273	2074522	2061797
Tiempo total de evolución	1979604	2083373	1537330	2064583	2052034
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(110,1)	(113,1)	(125,1)	(111,1)	(112,1)

Subgrafo de Requerimientos	F	G	H	I	J
Bytes promedio de individuo	1308	1308	1308	1308	1308
Bytes total de población	156960	156960	156960	156960	156960
Tiempo promedio de creación	63.275	60.6	63.3583	64.9583	58.6417
Tiempo total de creación	7593	7272	7603	7795	7037
Tiempo de cruza promedio	429.0465	425.0332	432.0764	414.2691	432.3522
Tiempo de cruza total	129143	127935	130055	124695	130138
Tiempo de mutación promedio	4924.548	4568.186	4796.455	4532.505	4757.339
Tiempo de mutación total	1482289	1375024	1443733	1364284	1431959
Tiempo total de ejecución	1658661	1550143	1621912	1536254	1609412
Tiempo total de evolución	1651041	1542844	1614283	1528433	1602348
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(115,1)	(119,299)	(121,1)	(113,1)	(113,1)

	Media	Desviación Estándar
Tiempo (seg)	1774	245
Costo	115.2	4.87

A3.3.3.5 Problema de Validación 8 (PV8)

Binaria	A	B	C	D	E
Bytes promedio de individuo	8076	8076	8076	8076	8076
Bytes total de población	969120	969120	969120	969120	969120
Tiempo promedio de creación	103.35	103.125	116.475	127.6083	3872.5
Tiempo total de creación	12402	12375	13977	15313	464700
Tiempo de cruza promedio	2849.631	3094.256	3204.133	3272.907	3110.382
Tiempo de cruza total	857739	931371	964444	985145	936225
Tiempo de mutación promedio	5034.11	4432.674	4697.163	4573.398	4587.774
Tiempo de mutación total	1515267	1334235	1413846	1376593	1380920
Tiempo total de ejecución	2386697	2279315	2393559	2378339	2783161
Tiempo total de evolución	2374248	2266858	2379536	2362980	2318416
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(96,261)	(96,281)	(101,181)	(100,252)	(102,84)

Binaria	F	G	H	I	J
Bytes promedio de individuo	8076	8076	8076	8076	8076
Bytes total de población	969120	969120	969120	969120	969120
Tiempo promedio de creación	106.6417	119.2583	56.3917	121.2833	2330.392
Tiempo total de creación	12797	14311	6767	14554	279647
Tiempo de cruza promedio	3181.395	3228.947	3085.983	3047.176	3075.588
Tiempo de cruza total	957600	971913	928881	917200	925752
Tiempo de mutación promedio	4545.472	4844.674	5030.326	4323.186	4562.791
Tiempo de mutación total	1368187	1458247	1514128	1301279	1373400
Tiempo total de ejecución	2339905	2445773	2451161	2234338	2580111
Tiempo total de evolución	2327037	2431411	2444338	2219732	2300417
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(98,282)	(103,202)	(97,281)	(93,112)	(93,195)

	Media	Desviación Estándar
Tiempo (seg)	2427	157
Costo	97.9	3.54

Lista de Caminos	A	B	C	D	E
Bytes promedio de individuo	8988.267	8777	8894.733	8910.934	8815.467
Bytes total de población	1078592	1053240	1067368	1069312	1057856
Tiempo promedio de creación	138.0333	142.775	154.075	210.4667	185.3583
Tiempo total de creación	16564	17133	18489	25256	22243
Tiempo de cruza promedio	58.3322	57.4452	59.2226	83.3854	91.0565
Tiempo de cruza total	17558	17291	17826	25099	27408
Tiempo de mutación promedio	12842.12	11623.81	12478.35	14853.18	16426.58
Tiempo de mutación total	3865477	3498766	3755982	4470807	4944401
Tiempo total de ejecución	3910391	3544023	3803315	4537445	5011442
Tiempo total de evolución	3893777	3526822	3784780	4512120	4989122
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(122,1)	(121,160)	(116,1)	(120,1)	(121,1)

Lista de Caminos	F	G	H	I	J
Bytes promedio de individuo	8853.134	8933.467	8666.533	8803.134	8849.6
Bytes total de población	1062376	1072016	1039984	1056376	1061952
Tiempo promedio de creación	185.3167	196.9667	130.8917	200.6917	151.6667
Tiempo total de creación	22238	23636	15707	24083	18200
Tiempo de cruza promedio	87.485	89.3223	87.6977	85.4452	57.3621
Tiempo de cruza total	26333	26886	26397	25719	17266
Tiempo de mutación promedio	16142.78	17663.7	9473.92	16214.06	12062.63
Tiempo de mutación total	4858977	5316773	2851650	4880431	3630851
Tiempo total de ejecución	4924123	5384307	2909933	4946311	3677113
Tiempo total de evolución	4901809	5360603	2894161	4922161	3658863
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(119,3)	(120,1)	(126,1)	(117,1)	(121,1)

	Media	Desviación Estándar
Tiempo (seg)	4265	804
Costo	120.3	2.75

Subgrafo de Requerimientos	A	B	C	D	E
Bytes promedio de individuo	2820	2820	2820	2820	2820
Bytes total de población	338400	338400	338400	338400	338400
Tiempo promedio de creación	148.9833	153.75	166.675	215.7917	192.9333
Tiempo total de creación	17878	18450	20001	25895	23152
Tiempo de cruza promedio	926.8439	862	899.5947	1003.522	1064.209
Tiempo de cruza total	278980	259462	270778	302060	320327
Tiempo de mutación promedio	15616.8	14302.06	14971.59	15094.28	16763.47
Tiempo de mutación total	4700657	4304921	4506450	4543378	5045805
Tiempo total de ejecución	5056547	4641646	4856479	4949416	5469282
Tiempo total de evolución	5038614	4623141	4836417	4923448	5446059
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(122,1)	(121,160)	(116,1)	(120,1)	(121,1)

Subgrafo de Requerimientos	F	G	H	I	J
Bytes promedio de individuo	2820	2820	2820	2820	2820
Bytes total de población	338400	338400	338400	338400	338400
Tiempo promedio de creación	192.4667	205.0917	138.975	170.5	161.9833
Tiempo total de creación	23096	24611	16677	20460	19438
Tiempo de cruza promedio	1055.708	1048.98	1087.216	844.0465	871.4385
Tiempo de cruza total	317768	315743	327252	254058	262303
Tiempo de mutación promedio	16465.44	18044.94	10258.77	15112.73	14524.59
Tiempo de mutación total	4956098	5431527	3087891	4548931	4371902
Tiempo total de ejecución	5375613	5851395	3509428	4880923	4712286
Tiempo total de evolución	5352453	5826720	3492685	4860414	4692796
Cantidad de generaciones	300	300	300	300	300
Mejor individuo (costo,iteración)	(119,3)	(120,1)	(126,1)	(117,1)	(121,1)

	Media	Desviación Estándar
Tiempo (seg)	4930	625
Costo	120.3	2.75

A3.3.3.6 Problema 1 con 2000 Iteraciones (PCm1)

En las siguientes tablas los tiempos están en milisegundos. Los carácteres de la A a la C representan diferentes secuencias de números aleatorios (entre estructuras, un mismo carácter no corresponde a la misma secuencia).

Binaria	A	B	C
Bytes promedio de individuo	19812	19812	19812
Bytes total de población	2971800	2377440	2971800
Tiempo promedio de creación	303.9333	486.625	438.72
Tiempo total de creación	45590	58395	65808
Tiempo de cruza promedio	133.1989	239.1035	191.1604
Tiempo de cruza total	266531	478446	382512
Tiempo de mutación promedio	5862.944	4579.346	5665.584
Tiempo de mutación total	11731751	9163272	11336833
Tiempo total de ejecución	12067069	9719267	11809538
Tiempo total de evolución	12021410	9660834	11743534
Cantidad de generaciones	2000	2000	2000
Mejor individuo (costo,iteración)	(906,1985)	(745,1555)	(966,341)

Lista de Caminos	A	B	C
Bytes promedio de individuo	1876.933	1883	1881.813
Bytes total de población	225232	225960	282272
Tiempo promedio de creación	58.6333	120.675	60.54
Tiempo total de creación	7036	14481	9081
Tiempo de cruza promedio	40.1929	37.6693	59.0575
Tiempo de cruza total	80426	67707	118174
Tiempo de mutación promedio	5501.718	6126.876	8772.917
Tiempo de mutación total	11008937	12133003	17554606
Tiempo total de ejecución	11145572	16208132	17745478
Tiempo total de evolución	11138432	12193600	17736344
Cantidad de generaciones	2000	2000	2000
Mejor individuo (costo,iteración)	(507,1014)	(527,999)	(485,1753)

Subgrafo de Requerimientos	A	B	C
Bytes promedio de individuo	660	660	660
Bytes total de población	79200	79200	79200
Tiempo promedio de creación	69.1167	64.7833	68.65
Tiempo total de creación	8294	7774	8238
Tiempo de cruza promedio	388.2139	402.5772	437.5614
Tiempo de cruza total	776816	805557	437999
Tiempo de mutación promedio	5913.674	6902.437	6969.612
Tiempo de mutación total	11833261	13811776	16976582
Tiempo total de ejecución	12771214	14774655	15513188
Tiempo total de evolución	12762846	14766837	15504950
Cantidad de generaciones	2000	2000	2000
Mejor individuo (costo,iteración)	(500,1562)	(592,178)	(485,1753)