Práctica 4

Metaheurísticas Basadas en Poblaciones

Metaheurísticas – Grado de Ingeniería Informática

Universidad de Córdoba

2015 / 2016

Grupo: NPGroup

Miembros:

Daniel Carmona Martínez
Cristian García Jiménez
Antonio Jesús Jiménez Urbano
Guillermo Sugráñez Pérez

Índice

Índice de Grafos

Índice de Gráficas

Introducción

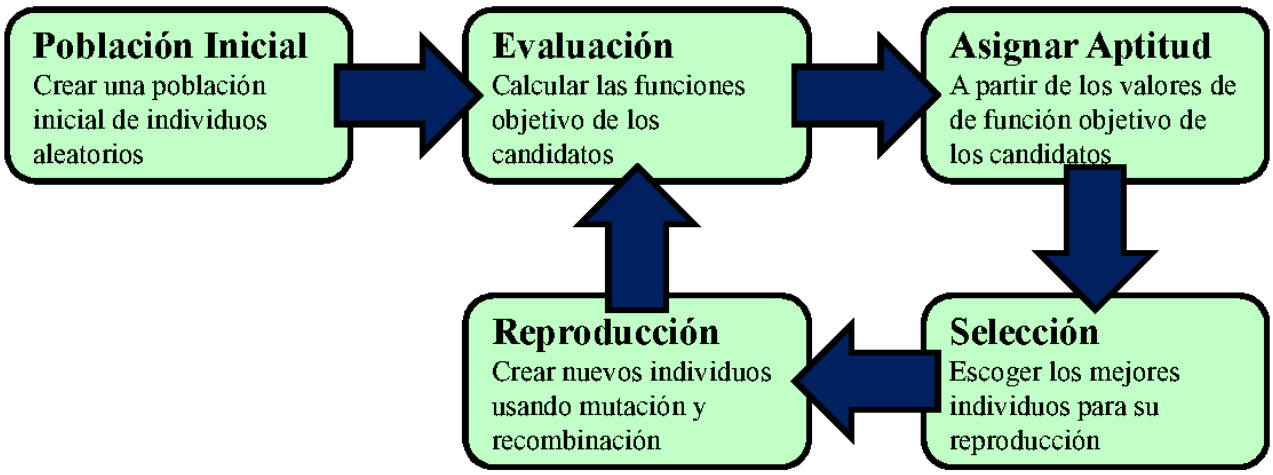
En esta práctica se ha abordado nuevamente la resolución de los problemas de la mochila y el viajante de comercio (vistos anteriormente en la práctica 1, 2 y 3), pero empleando un algoritmo evolutivo con el objetivo de cuantificar el fitness en función de las distintas poblaciones generadas en cada evaluación.

Este documento se divide en cuatro secciones diferentes. Una primera sección donde se explican los conceptos previos, una segunda donde se detallan los aspectos de la codificación de este nuevo algoritmo, una tercera donde se evalúan las pruebas realizadas, una cuarta donde se listan las conclusiones obtenidas a partir de dichas pruebas y finalmente una quinta donde se continua con el estudio de nuestro problema (Máxima Diversidad) en base a los estudios realizados en esta práctica.

Capítulo 1 - Consideraciones previas

1.1 Definición Algoritmo Evolutivo

Los algoritmos evolutivos son métodos de optimización y búsqueda de soluciones inspirados en los procesos de evolución natural y genética. En ellos se mantiene un conjunto de individuos (poblaciones) que representan posibles soluciones, las cuales se mezclan, y compiten entre sí, de tal manera que las más aptas son capaces de prevalecer a lo largo del tiempo, evolucionando hacia mejores soluciones cada vez. El esquema de funcionamiento del algoritmo evolutivo consta de las siguientes fases:



1.2 Nomenclatura

A continuación, se especificarán los términos utilizados a lo largo del documento:

- **Genotipo:** Manera de representar una solución.
- **Alelos:** Cada uno de los elementos que componen el genotipo.
- **Individuo**: Posible solución del problema formada por alelos.
- **Población**: Conjunto de individuos en una determinada generación.
- **Generación**: Lapso temporal entre la combinación de una determinada población

para generar otra.

- **Padres**: Individuos que se reproducen para generar nuevos individuos (**hijos**) que formarán parte de la siguiente población.
- **Fenotipo**: Manifestación o significado del genotipo en cada problema en concreto.

Capítulo 2 - Codificación

```
std::vector<Solucion> Genetico(const int & numPadres,const int
  & epocas,const int & numContendientes,const int & alelos)
```

Para este algoritmo se han implementado las siguientes funciones:

- **Inicialización de la Población Inicial.**

  Genera los individuos que formarán parte de la primera población de manera aleatoria .

- **Evaluar Población Actual.**

  Recorre los individuos y determina cuál de ellos es el mejor de dicha población y al mismo tiempo comprueba si mejora la mejor solución obtenida hasta el momento en las diferentes generaciones.

- **Seleccionar Padres.**

  Se celebrán torneos con n padres aleatorios y se elige el mejor de éstos.

- **Operador de cruce.**

  Es el encargado de mezclar los padres de una nueva población de una forma determinada según el problema.

- **Generar Nueva Población**

  Reúne individuos para crear una nueva población con diferentes criterios.

- **Operador de mutación.**

  Altera los individuos de la población actual de forma aleatoria.

Estas funciones se explicarán con detalle en la sección correspondiente a cada uno de los problemas.

2.1 Problema de la mochila (KP)

Para este problema el **genotipo** sería un vector binario, los **alelos** representan los materiales y su **fenotipo** son los materiales que entran en la solución.

Se han realizado dos implementaciones del algoritmo, una basada en **maximizar el beneficio** de la solución controlando siempre que esta solución no exceda el tamaño de la mochila y otra basada en **minimizar la distancia** que existe entre el peso actual y la capacidad máxima de la mochila (sin sobrepasarla).

2.1.1 Maximizar Beneficio

- **Inicialización de la Población Inicial.**

  Se generan de forma heurística soluciones válidas (no sobrepasan la capacidad de la mochila).

- **Evaluar Población Actual.**

  Elige el mejor individuo de la población; Aquel que, sin superar el tamaño de la mochila, maximiza el beneficio obtenido

- **Seleccionar Padres.**

  Se realiza un torneo con N contendientes, eligiéndose el mejor de entre estos N individuos y devolviendo el resto para siguientes torneos. El padre ya seleccionado no puede volver a entrar en posteriores contiendas. De este modo se evita repetir padres.

- **Operador de cruce.**

  Operador UX(Uniform Crossover) aleatorizado.

  Este operador mezcla dos padres eligiendo K alelos del primero y añadiendo los restantes del segundo. Para su correcta implementación se genera una mascara binaria que representa la selección de alelos del primer padre (mascara a 1) y la selección de alelos del segundo (mascara a 0), un ejemplo seria el siguiente:

Alelo → 1 2 3 4 5 6 7

Mascara → 0 1 0 1 0 0 1

Padre 1 → 0 0 1 0 1 0 1

Padre 2 → 1 0 0 0 1 0 1

En este caso, se eligen los alelos 2, 4 y 7 del padre 1, y los alelos 1, 3, 5, 6 del padre 2, obteniendo el hijo:

Mascara → 0 1 0 1 0 0 1

Padre 1 → _ 0 _ 0 _ _ 1

Padre 2 → 1 _ 0 _ 1 0 _

Hijo → 1 0 0 0 1 0 1

Esto se repite para todos los padres seleccionados, cruzando cada uno de ellos con el resto y generando, siendo N el numero de padres, (N*N)-N hijos .

- **Generar Nueva Población**

  Para evitar soluciones que excedan la capacidad de la mochila, se eligen de entre los hijos generados en el cruce anterior aquellos que no excedan el tamaño de esta. Los restantes para mantener el numero de individuos estable (De ser necesario completar el tamaño de población) se eligen de forma aleatoria de entre los de la población actual aquellos que no excedan la mencionada capacidad. De no conseguir aún individuos suficientes para mantener el tamaño mencionado, se eligen estos de los que queden aunque excedan la capacidad maxima.

- **Operador de mutación.**

  Se realiza una inversión de bit en un alelo escogido de forma aleatoria para cada nuevo hijo generado. Para evitar una excesiva mutación a medida que avanzan las épocas, se ha introducido una probabilidad de mutación consiguiendo una mayor diversidad al inicio. Dado un determinado número de evaluaciones se vuelve a aumentar la probabilidad de mutación.

2.1.2 Minimizar Distancia

- **Inicialización de la Población Inicial.**

  Se genera de forma aleatoria. La probabilidad de introducir un 1 o un 0 en cada alelo es del 0’5.

- **Evaluar Población Actual.**

  Elige el mejor individuo de esta población que no supere la capacidad de la mochila y, además, comprueba si este mismo mejora el coste de la mejor solución global.

- **Seleccionar Padres.**

  Se escogen los contendientes que participarán en el torneo, se calculan sus

distancias a la capacidad máxima de la mochila **(*tanto si se pasa como si no*)** y se escoge el que posea un menor valor de esta distancia (en valor absoluto). Un padre ya seleccionado en una de estas operaciones no puede volver a ser seleccionado.

- **Operador de cruce.**

  Se ha implementado uno propio en el cual se mezclan todos los padres generados en la función anterior, cada nuevo hijo obtenido a partir de cada par de éstos posee una probabilidad del 0,5 de heredar cada alelo perteneciente al primer padre o al otro. Si el nuevo alelo heredado hace que se sobrepase la capacidad máxima de la mochila éste se transforma automáticamente a 0 (el material no entra en la solución).

- **Generar Nueva Población**

  Para la nueva población, se introducen todos los nuevos hijos generados (si caben) y si aun así no se llega a cubrir el tamaño máximo de ésta se introducen de forma aleatoria individuos de la población actual. Por último, se intercambia de forma aleatoria un individuo de la población siguiente por el mejor individuo de la actual (mayor coste y no sobrepasa la capacidad máxima).

- **Operador de mutación.**

  Se realiza de igual forma que en la implementación anterior.

2.2 Problema del Viajante de Comercio (TSP)

Para este problema el **genotipo** sería un vector de permutaciones, cada **alelo** representa un nodo o ciudad y su **fenotipo** la configuración para un determinado camino.

- **Inicialización de la Población Inicial.**

  Para este caso, se inicializa cada vector de manera ascendente y posteriormente se permuta aleatoriamente.

- **Evaluar Población Actual.**

  Para evaluar la población, se busca el candidato que minimice la distancia de entre todos los actuales. Una vez encontrado se compara con el óptimo global para comprobar si lo mejora.

- **Seleccionar Padres.**

  Se realiza un torneo con n individuos aleatorios y se selecciona el que posea menor coste y no sea padre. Esta operación se realiza tantas veces como padres quieran conseguirse.

- **Operador de cruce.**

  Operador PMX(Partially Mapped Crossover) aleatorizado.

  Al igual que en el UX, este operador mezcla dos padres eligiendo K alelos del primero y añadiendo los restantes del segundo. Su peculiaridad es que esta diseñado para la mezcla de soluciones cuyas representaciones están basadas en permutaciones sin repetición de N elementos. Al igual que en el operador mencionado,se genera una mascara binaria que representa la selección de alelos del primer padre (mascara a 1) y la selección de alelos del segundo (mascara a 0), un ejemplo seria el siguiente:

Alelo → 1 2 3 4 5 6 7

Mascara → 0 1 0 1 0 0 1

Padre 1 → 5 3 2 0 6 1 4

Padre 2 → 1 0 4 5 3 6 2

En este caso, se eligen los alelos 2, 4 y 7 del padre 1, y los alelos 1, 3, 5, 6 del padre 2, obteniendo el hijo:

Mascara→ 0 1 0 1 0 0 1

Padre 1 → _ 3 _ 0 _ _ 4

Padre 2 → 1 0 4 5 3 6 2

Para generar el hijo, se introducen en este los alelos del padre 1:

Hijo → _ 3 _ 0 _ _ 4

A continuación se van introduciendo de forma ordenada los alelos del padre 2, comprobando que dichos alelos no se encuentren ya en el hijo. En dicho caso se descarta ese alelo y se introduce el siguiente que no se encuentre ya el nuevo individuo:

- Se introduce el valor 1

Padre 2 → _ 0 4 5 3 6 2

Hijo → 1 3 _ 0 _ _ 4

- Se descarta el valor 0, ya esta en la solución

Padre 2 → _ _ 4 5 3 6 2

Hijo → 1 3 _ 0 _ _ 4

- Se descarta el valor 4, ya esta en la solución

Padre 2 → _ _ _ 5 3 6 2

Hijo → 1 3 _ 0 _ _ 4

- No se mostrarán más pasos intermedios para no alargar la explicación:

Padre 2 → _ _ _ _ _ _ _

Hijo → 1 3 5 0 6 2 4

Esto se repite para todos los padres seleccionados, cruzando cada uno de ellos con el resto y generando, siendo N el numero de padres, (N*N)-N hijos.

- **Generar Nueva Población**

  Para la nueva población, se introducen todos los nuevos hijos generados (si caben) y si aun así no se llega a cubrir el tamaño máximo de ésta se introducen de forma aleatoria individuos de la población actual. Por último, se intercambia de forma aleatoria un individuo de la población siguiente por el mejor individuo de la actual .

- **Operador de mutación.**

  Se intercambian de forma aleatoria dos alelos de cada hijo generado. Dado un determinado número de evaluaciones se vuelve a aumentar la probabilidad de mutación.

Capítulo 3 - Evaluación

El estudio realizado consta de diez experimentos independientes para cada instancia del problema en cuestión. Para cada problema se ha elegido un número diferente de generaciones dependiendo del número de elementos en la solución y, a su vez, se ha limitado el tiempo de ejecución a 10 minutos para prevenir que éste se demore en exceso.

En este apartado se mostrarán las gráficas y tablas correspondientes a las diferentes experimentos realizados junto con sus correspondientes conclusiones:

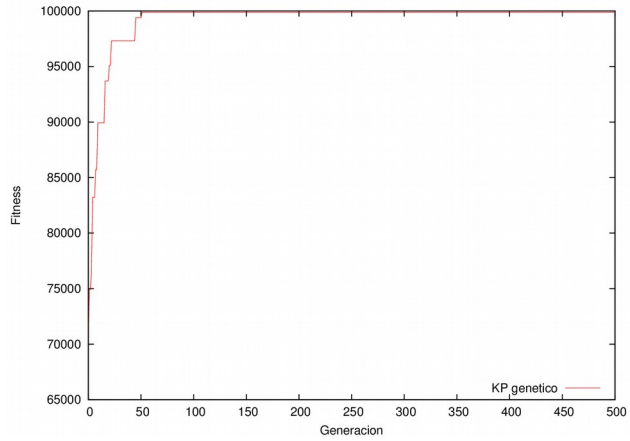
3.1 Problema de la mochila (KP)

3.1.1 Maximizar Beneficio vs Minimizar Distancia

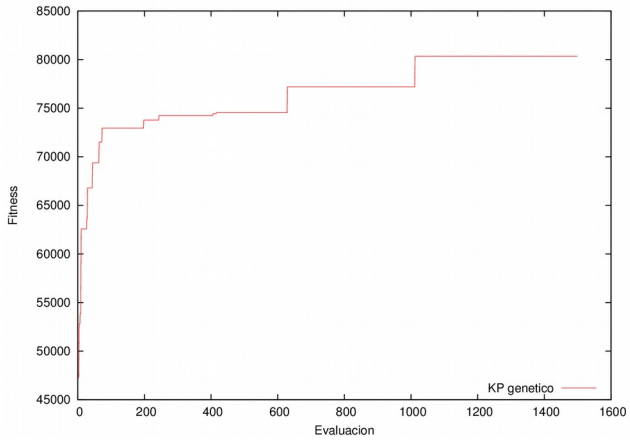
A la izquierda se muestran la gráficas correspondientes a la primera implementación (Maximizar Beneficio) y junto a ella las correspondientes a la segunda (Minimizar Distancia) con el fin de comparar cada criterio.

Nota: Se usan diferentes escalas para poder representar bien las gráficas ya que cada uno converge de forma distinta.

- **KP200**

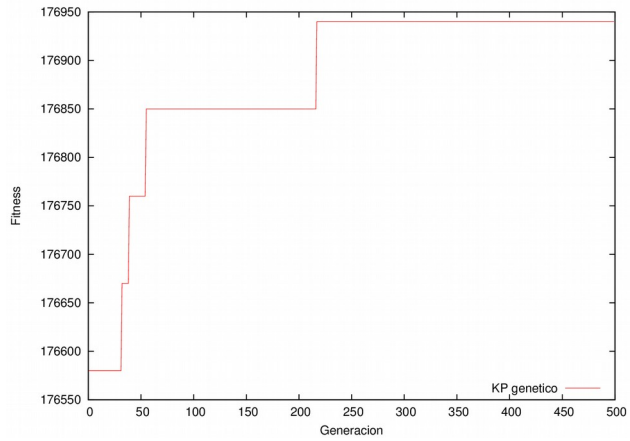


Gráfica 1: knapPI_200

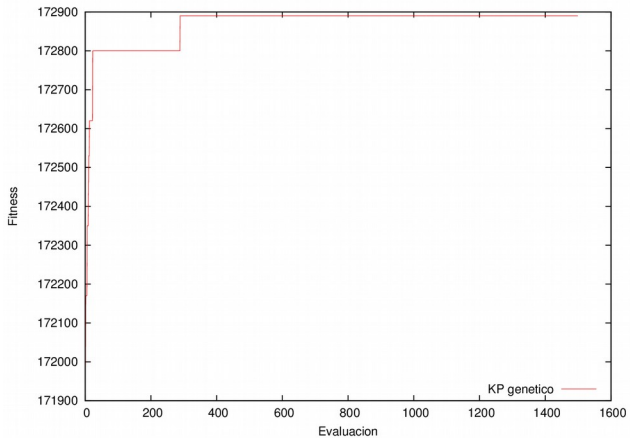


Gráfica 2: knapPI_200

- **KP500**

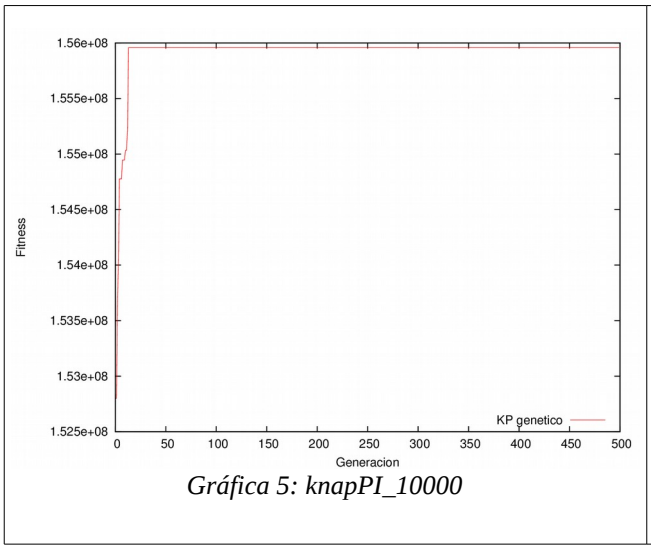


Gráfica 3: knapPI_500

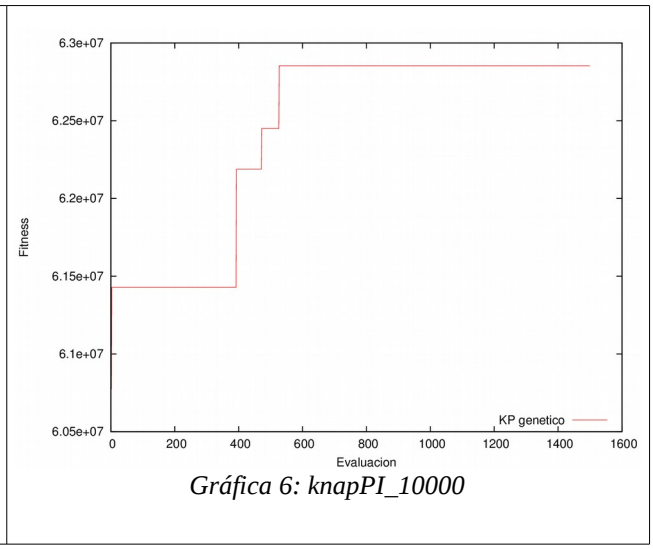


Gráfica 4: knapPI_500

- **KP10000**



Gráfica 5: knapPI_10000



Gráfica 6: knapPI_10000

3.1.2 Conclusiones

Podemos observar que a la hora de **maximizar el beneficio** obtenemos **mejores resultados** en todos los casos, además de converger más rapidamente (excepto en el caso de KP500).

Puede observarse que llegado a un determinado óptimo aunque se aumente la mutación tras un determinado número de evaluaciones ni siquiera esto consigue mejorar la mejor solución obtenida. Esto es debido a que el algoritmo converge con mucha velocidad evaluando en bastantes casos varias generaciones que no aportan mejora en la solución.

- **Maximizar beneficio (200):**

1	2	3	4	5	6	7	8	9	10
85489,496	83764,116	94680,898	85098,696	92849,03	89249,486	83260,358	89142,914	88548,224	99175,802
									Media global: 89125,902

- **Minimizar distancia (200):**

1	2	3	4	5	6	7	8	9	10
76456,4756	72277,9686	57877,6717	57942,3328	57656,8278	55590	55590	51449	68805,4823	47732
									Media global: 60137,7759

Ratio maximizar beneficio frente minimizar distancia:

89125,902 / 60137,7759 = 1,5.

Por tanto, maximizar beneficio es 1,5 veces mejor que minimizar distancia.

- **Maximizar beneficio (500):**

1	2	3	4	5	6	7	8	9	10
176794,38	176878,26	176741,28	176733	176770,08	176780,88	176938,56	176630,22	176655,96	176681,52
Media global: 176760,414									

- **Minimizar distancia (500):**

1	2	3	4	5	6	7	8	9	10
172437,66	172161,42	172680,06	172866,9	172885,44	172791,78	172794,54	173244,72	173152,68	172067,88
Media global: 172708,308									

Ratio maximizar beneficio frente minimizar distancia:

176760,414 / 172708,308 = 1,02.

Por tanto, maximizar beneficio es 1,02 veces mejor que minimizar distancia.

- **Maximizar beneficio (10000):**

1	2	3	4	5	6	7	8	9	10
153287180	143104490	148290118	145350044	141932454	149206710	155919484	146013142	147253792	146693114
Media global: 147705052,8									

- **Minimizar distancia (10000):**

1	2	3	4	5	6	7	8	9	10
4957563083,6	59503264,6	58116484,4	59038361,3	62082222,3	59805045	62428882	60461878	61641985,6	59531608,2
Media global: 55001728,153									

Ratio maximizar beneficio frente minimizar distancia:

147705052,8 / 55001728,153 = 2,69.

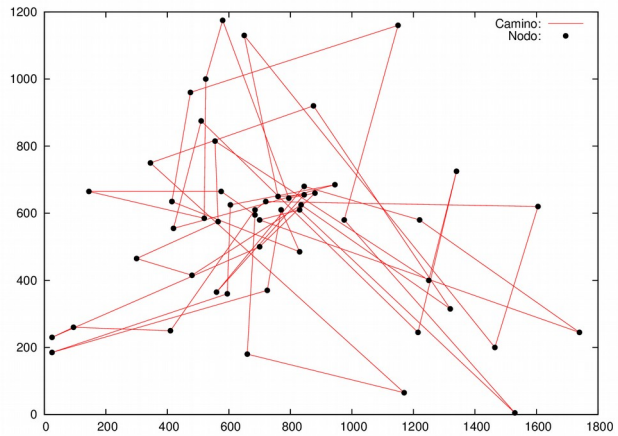
Por tanto, maximizar beneficio es 2,69 veces mejor que minimizar distancia.

Lo que se quería comprobar al introducir estas dos implementaciones era si sería mejor intentar minimizar el espacio vacío en la mochila frente a maximizar el beneficio obtenido por los materiales. Como conclusión se puede afirmar que buscar un beneficio mayor es mejor estrategia.

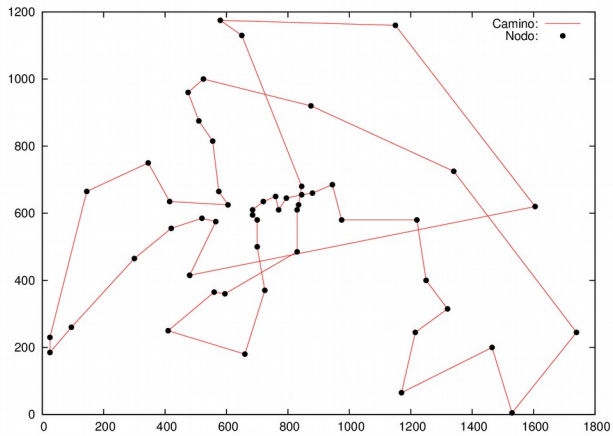
3.2 Viajante de comercio (TSP)

3.2.1 Grafos

- **Berlín52**

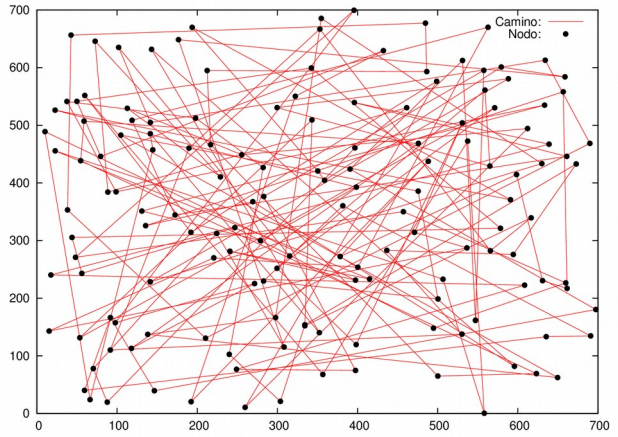


Grafo 1: Berlin52. Solución inicial.

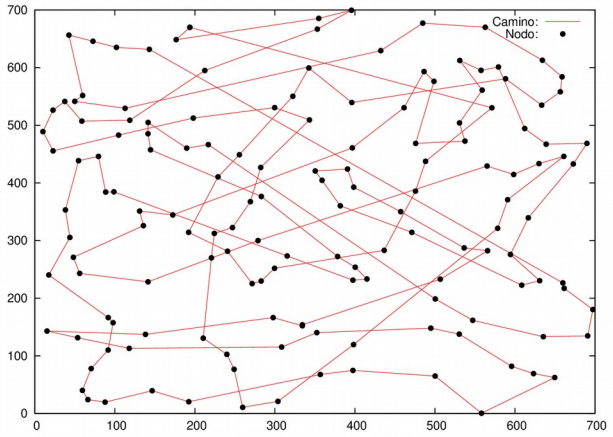


Grafo 2: Berlin52. Solución Óptima.

- **CH150**



Grafo 3: CH150. Solución Inicial.



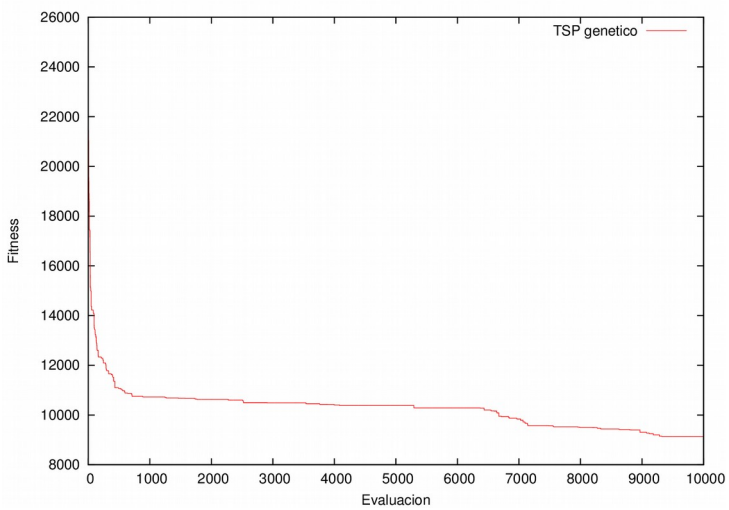
Grafo 4: CH150. Solución Óptima.

- **D2103**

Para la instancia d2103 no se van a incluir grafos. Esto se debe a que en el tiempo considerado (10 minutos) y debido al tamaño del problema solo se evaluan 64 generaciones y no son suficientes para observar cambios significativos en el grafo.

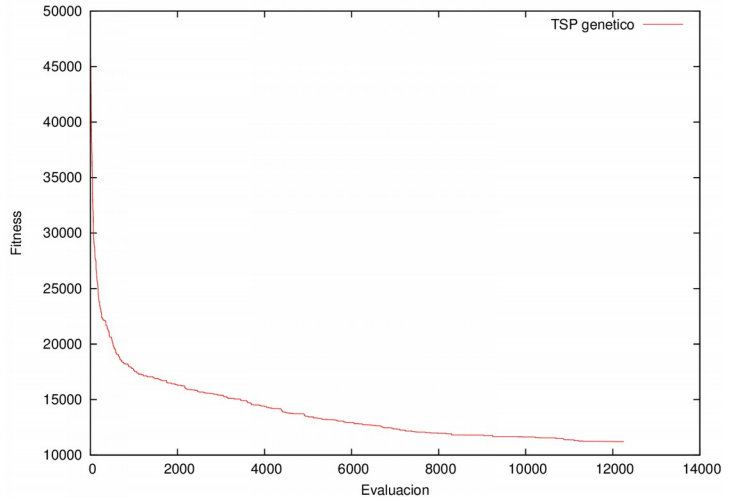
3.2.2 Gráficas

- **Berlín52**



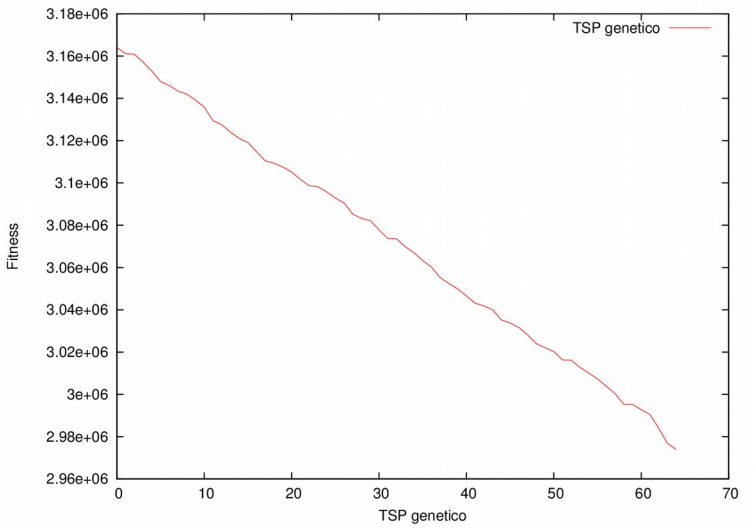
Gráfica 7: Berlin52

- **CH150**



Gráfica 8: ch150

- **D2103**



Gráfica 9: d2103

3.2.3 Conclusiones:

Se puede observar que debido a la aleatoriedad del operador de cruce se necesitan muchas más evaluaciones para llegar a un óptimo. Aun así, puede apreciarse que esta aleatoriedad unida a la presión selectiva que ejerce el operador de selección y la introducción del mejor individuo de la población actual en la población siguiente hace que se consigan muy buenos resultados en un número adecuado (10000 para berlin52 y 30000 para ch150).

Capítulo 4 - Conclusiones Generales

De la gran cantidad de pruebas necesarias para obtener los parámetros óptimos de ejecución parece lógico pensar que la obtención de estos es en sí otro problema que debería resolverse antes de realizar las pruebas. También podemos concluir que los algoritmos genéticos son muy eficientes a la hora de conseguir un valor óptimo pero en función de los diferentes operadores que se implementen el tiempo de ejecución puede llegar a ser realmente tedioso. Además, la optimización de estos operadores es otro de los retos a asumir a la hora de implementarlos.

Capítulo 5 - Problema de la diversidad máxima.

De esta práctica lo que hemos decidido reutilizar para nuestro problema han sido los operadores de evaluación y cruce empleados para el problema del viajante de comercio. Además se están estudiando el fenotipo y genotipo para representar las diferentes soluciones puesto que la forma en la que se representan en los problemas abordados en prácticas no nos aportan una forma correcta para interpretar el problema de la máxima diversidad.

Bibliografía

GENDREAU, Michel. 2010. *Handbook of Metaheuristics: International Series in Operations Research & Management Science. Jean-Yves Potvin .2ª Edición. New York: Springer. 978-1-4419-1663-1*