Inteligencia Computacional
Master en Ingeniería Informática

Práctica 2: Algoritmos Genéticos

Luis Alberto Segura Delgado

Viernes 29 de Enero de 2016

Índice

1 Introducción

En esta práctica el objetivo es familiarizarse y comprender el funcionamiento de los **algoritmos evolutivos**, en concreto, los **algoritmos genéticos**. En la siguiente sección se explicarán cada una de las variantes de algoritmos genéticos que se han implementado para la práctica.
El problema a resolver es la asociación cuadrática (Quadratic Assignment Problem; QAP). Este problema consiste en asignar una serie de localizaciones a una serie de elementos, en nuestro caso, instalaciones. El objetivo es construir esas instalaciones en los lugares minimicen el coste de transporte de materiales. Entre las instalaciones, una vez construidas, se debe transportar un material determinado, y sabemos el coste llevar dicho material de una instalación a otra según donde se coloque.

La función de coste (función a minimizar y, por tanto, la función *fitness*) es la siguiente:

$$\sum_{i,j} w(i,j)d(p(i),p(j)) \tag{1}$$

donde p() define una permutación sobre el conjunto de instalaciones (la permutación que representa una la solución al problema). $w(i,j)$ indica el peso (coste) asociado a llevar material de la instalación i a la j; y $d(x,y)$ indica la distancia entre las instalaciones x e y.

Una vez que conocemos nuestro problema, vamos a ver los algoritmos propuestos para resolverlo.

2 Implementación

En esta sección se describen las diferentes variantes de algoritmos genéticos propuestos e implementados para resolver el problema de la asignación cuadrática. En primer lugar se describe la representación de la solución y los operadores básicos, ya que son comunes a todas las implementaciones. Y finalmente se describen brevemente los algoritmos concretos.

2.1 Representación de la solución

La representación de la solución es sencilla y una de las más usadas para el QAP. Se hace uso de una representación en forma de permutación. Un vector de valores enteros en el que la posición del vector indica la localización y el contenido de esa posición representa el instalación que ocupará esa localización.

2.2 Operadores

2.2.1 Operador de Cruce (OX)

El operador de cruce elegido e implementado para nuestros algoritmos ha sido el **OX**. Existen más operadores de cruce para utilizar, pero me he decantado por este porque parece dar buenos resultados y, personalmente, me gusta más.

Este operador consiste en cambiar el orden de los padres en los hijos y que hereden la parte central del cromosoma. En primer lugar se elige un rango central de elementos del cromosoma que se mantendrán de los padres a los hijos. El resto de posiciones a los extremos de los hijos se rellenan con la información del otro padre. Para ello, se van rellenando las posiciones de principio a fin en el mismo orden que en el padre. Es decir, si la posición 0 del cromosoma del padre es un 4, se intenta poner en la posición 0 del hijo un 4. En caso de que el 4 ya este en el centro del cromosoma que recibió del otro padre, se salta a la siguiente posición del padre y se pone el siguiente elemento. De esta forma mantenemos el orden.

Básicamente este operador de cruce permite que los hijos reciban la parte intermedia del cromosoma de uno de los padres, y el orden de los elementos de los extremos del otro padre. En la figura 1 podemos ver una representación gráfica de como obtener un hijo a partir de dos padres.

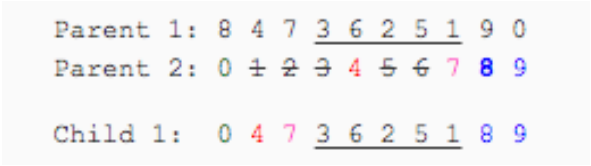

Figura 1: Operador de Cruce OX

2.2.2 Operador de Selección

Para la selección de individuos a cruzar, se ha elegido un operador de selección muy simple. La selección es completamente aleatoria. En cada generación se eligen parejas de padres aleatoriamente y por cada pareja de padres se generan dos hijos que reemplazan a sus padres en la población. Una vez reemplazada toda la población por los nuevos individuos (hijos), se aplica la mutación (si se da el caso) y se vuelve a empezar.

2.2.3 Operador de Mutación

El operador de mutación implementado muta los dos hijos generados a partir de los padres con el operador de cruce. De forma aleatoria se elige si se mutará cada uno de los hijos con una probabilidad de mutación. La probabilidad de mutación es un parámetro de nuestros algoritmos, así que puede modificarse para tratar de mejorar las soluciones.

La mutación que se aplica a los individuos consiste en un intercambio de genes. Se eligen aleatoriamente dos genes, y se intercambian. En nuestro problema, se eligen aleatoriamente dos localizaciones, y se intercambian las instalaciones asignadas a dichas localizaciones.

2.3 Técnica de Optimización Local

Para las variantes baldwiniana y lamarckiana del algoritmo genético, se ha optado por utilizar una **búsqueda local** que permita optimizar al máximo las soluciones de nuestra población, hasta llegar a un mínimo local. La búsqueda local que se ha implementado recibe una solución y la optimiza buscando el primer vecino mejor que ésta. Es decir, en cuanto obtenemos un vecino mejor que nuestra solución actual, "nos movemos" al vecino e iniciamos una nueva búsqueda local sobre la nueva solución, buscando el primer vecino a esta nueva solución que sea mejor. Si recorremos todo el vecindario y no se ha encontrado ningún vecino mejor, se acaba la búsqueda y devuelve el mejor encontrado. La condición de parada es no encontrar ningún vecino mejor que la solución actual (estamos estancados en un mínimo local), o superar el número de evaluaciones máximo (es un parámetro).

2.3.1 Operador de Vecino

El operador de vecino nos indica como nos movemos por el vecindario de una solución, y por tanto, cual es el vecindario de la misma.
El operador de vecino utilizado es el mismo que el operador genético, consiste en intercambiar dos instalaciones de diferentes localizaciones. En lugar de elegir dos localizaciones aleatoriamente, en la búsqueda local se intercambian todas con todas. El vecindario para una solución esta formado por todas las soluciones que podemos obtener intercambiando todas las instalaciones de localización. Si después de probar todas

las combinaciones no hemos encontrado una solución vecina mejor o superamos el máximo de evaluaciones, tendremos que finalizar la búsqueda local.

2.4 Algoritmo Genético Estándar

En primer lugar se ha implementado un algoritmo genético básico, que utiliza los operadores de selección, cruce y mutación explicados anteriormente.
Este algoritmo permite, al igual que los dos siguientes, que se indiquen un número de generaciones. El algoritmo acabará después de un número de generaciones especificado.

2.5 Algoritmo Genético variante Baldwiniana

Esta variante solicitada en la práctica es idéntica a la versión estándar, pero con la diferencia de que se aplica una optimización local (aplicando la búsqueda local explicada anteriormente) a cada generación. La optimización local se aplica una vez obtenida la nueva generación (a partir de los hijos) a todos los individuos. Una vez aplicada, se guarda el mejor individuo obtenido después de la optimización. Las soluciones optimizadas **no** se usan para generar la siguiente generación.

2.6 Algoritmo Genético variante Lamarckiana

Esta variante solicitada en la práctica es idéntica a la versión estándar, pero con la diferencia de que se aplica una optimización local (aplicando la búsqueda local explicada anteriormente) a cada generación. La optimización local se aplica una vez obtenida la nueva generación (a partir de los hijos) a todos los individuos. Una vez aplicada, se guarda el mejor individuo obtenido después de la optimización. Las soluciones optimizadas, en este caso, **si** se usan para generar la siguiente generación. Ésta es la única diferencia con respecto a la versión Baldwiniana.

3 Resultados

Ahora que conocemos las diferentes variantes de los algoritmos que hemos utilizado, vamos a comentar los resultados obtenidos para las diferentes versiones del problema QAP.

Tabla 1: Resultados sobre algunas bases de datos

	GA Básico	**GA Baldwiniano**	**GA Lamarckiano**
bur26a	5523345	5569408	5443330
bur26b	3886608	3924237	3826883
chr12a	15786	13914	9552
chr12b	16186	14700	9742
chr25a	11060	11626	5476
lipa20a	3842	3849	3701
lipa20b	32863	32921	27076
nug12	620	638	582
nug20	2970	2880	2602
tai60a	8253832	8217080	7804916

En la tabla 1 podemos ver algunos resultados obtenidos al ejecutar nuestras tres variantes de algoritmos genéticos sobre algunas de las bases de datos que se proporcionan en la práctica. Para estos experimentos se han utilizado los mismos parámetros para los tres algoritmos (Ver tabla 2).

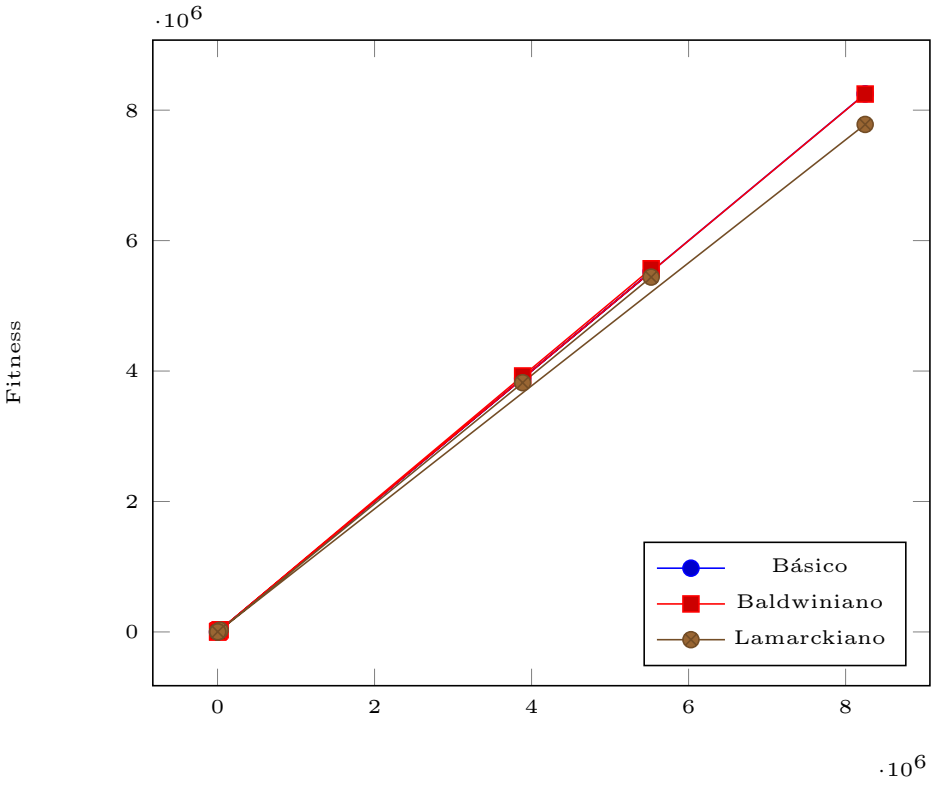

Figura 2: Gráfica resultados de las diferentes versiones

Tabla 2: Parámetros algoritmos

Prob Mutación	**Tamaño Población**	**Generaciones**	**Evaluaciones Busq. Local**
0.3	50	1000	1000

Como podemos ver en la tabla 1, el algoritmo que mejores resultados produce en todos los problemas es la variante Lamarckiana. Después encontramos la versión Baldwiniana y la básica. Estas dos últimas obtienen resultados muy similares entre ellas. Dependiendo del problema una será mejor que la otra, pero son muy similares. En la figura 2 podemos ver una gráfica que nos muestra la variación de la función fitness para cada problema. Como vemos, la versión Lamarckiana se encuentra por debajo de los otros dos en todos los problemas. Mientras que las otras dos versiones obtienen prácticamente los mismos resultados (por eso solo se aprecia el resultado de la versión Baldwiniana).

Una vez realizados estos primeros experimentos, llega el momento de enfrentarse con la base de datos más grande y para la que se piden resultados en la práctica, **tai256c**.

4 Conclusiones

C

Para concluir, decir que el código y esta documentación estarán disponibles en Github por si son de utilidad para alguien. La dirección es `https://github.com/segura2010/QAP-GeneticSolver-IC` y estará disponible pocos días después de la entrega de la práctica.