ugr | Universidad de Granada

Inteligencia Computacional

Práctica de algoritmos evolutivos
Problemas de optimización combinatoria
QAP

Realizado por:

Francisco Solano López Rodríguez

Máster universitario en Ingeniería Informática

1. Introducción

Para resolver el problema de la asignación cuadrática se han implementado 3 algoritmos. El primer algoritmo se corresponde con una algoritmo genético básico, mientras que los otros dos se corresponden con variantes de este. Uno de ellos se trata de una variante baldwiniana y el otro una variante lamarckiana.

Para las variantes se ha implementado un algoritmo de búsqueda local, conocido como primero el mejor, que consiste en explorar el vecindario de la solución y tomar el primer vecino que mejore a la solución actual y continuar el proceso hasta llegar a un criterio de parada o hasta que no se encuentre ninguna solución vecina que mejore la acutal.

1.1. Estructura del cromosoma y funciones básicas para el algoritmo genético

Los cromosomas en nuestro problema van a ser estructuras de datos con dos atributos. El primer atributo se va a corresponder con un vector de enteros, en el cual se va a almacenar una solución, que se corresponde con una permutación, a dicho vector lo vamos a representar con una **s**. El segundo atributo que va a almacenar un cromosoma es el fitness, es decir el valor de la función objetivo para la solución **s**.

Cruce

El cruce entre dos cromosomas va a consistir en lo siguiente:

1. Se van a tomar aquellas características en las que los valores de ambos padres coincidan y se van a colocar en el hijo en la misma posición en las que estaban. Este cruce conservan las características prometedoras, ya que si se repiten en ambos padres es problable que sean más prometedoras. Ejemplo:

```
1 padre1 {3,1,4,2,5,6}
2 padre2 {2,1,4,6,5,3}
3 hijo   {-,1,4,-,5,-}
```

2. Los valores que falten se añaden de forma aleatoria.

Mutación

Una mutación de un gen va a consistir simplemente en intercambiar el valor de dicho gen, con el de otro, es decir, intercambiar dos valores del vector solución.

Selección de padres

Para la selección de un padre para el cruce, se recurrirá a un torneo binario, el cual consiste en elegir dos individuos distintos de la población de forma aleatoria y quedarnos con el mejor de ellos.

Inicialización de la población

Lo primero que haremos en los algoritmos genéticos será inicializar la población, para ello generaremos individuos de forma aleatoria. Para asegurarnos de que los individuos contienen soluciones validas lo que se hará cada vez que se genere un individuo aleatorio, sea barajar aleatoriamente un vector de índices.

1.2. Algoritmo genético básico

El algoritmo genético básico que se ha implementado, consiste en un algoritmo genético generacional. El reemplazamiento se realiza con elitismo, es decir, se mantiene siempre al mejor individuo, el cual se sustituye por el peor de la generación.

El siguiente pseudocódigo muestra la lógica de la implementación realizada.

```
1  Funcion Generacional(max_generaciones)
2      tam_p = 50
3      mejor = 0, peor = 0
4
5      generarPoblacion(poblacion, tam_p, num_atrib)
6
7      Para i = 1 hasta tam_p-1
8          Si poblacion[i].valor > poblacion[mejor].valor
9              mejor = i
10
11     Para i = 1 hasta max_generaciones
12         mejor_cromosoma_old = poblacion[mejor]
13
14         Para j = 0 hasta tam_p
15             Si aleatorio ∈ [0,1]) < 0.7
16                 padre1 = torneoBinario(poblacion)
17                 padre2 = torneoBinario(poblacion)
18                 hijo = cruce(padre1, padre2)
19
20                 Para k = 0 hasta num_atrib-1
21                     Si (aleatorio ∈ [0,1] <= 0.001)
22                         Mutar(hijo.s_i)
23
24                 poblacion[j] = hijo
25
26             Si poblacion[j].valor > poblacion[mejor].valor
27                 mejor = j
28             Si poblacion[j].valor < poblacion[peor].valor
29                 peor = j
30
31         poblacion[peor] = mejor_cromosoma_old
32
33     Devolver poblacion[mejor].s
34 Fin
```

Podemos ver algunos de los parámetros que se han fijado. Por ejemplo la probabilidad de cruce es de 0.7 y la probabilidad de mutación por cada gen es de 0.001. Se han fijado estos valores como consecuencia de ensayo y error.

1.3. Algoritmo genético baldwiniano

Para esta variante se ha incorporado al algoritmo genético básico, la técnica de búsqueda local primero el mejor. En este caso se utiliza dicha búsqueda local para obtener el fitness del individuo pero sin utilizar las características aprendidas a la hora de cruzar dos cromosomas. Esta ténica se aplica cada 10 generaciones.

1.4. Algoritmo genético lamarckiano

Este algoritmo es similar al anterior, solo que en este caso si que se utilizan las características aprendidas para el cruce de los cromosomas, es decir, en este caso si se conservan las características aprendidas.

Nota: para las dos variantes se ha hecho uso de una búsqueda local de primero el mejor, donde se explora el vecindario de una solución y nos quedamos con la primera solución que mejore la actual.

Los vecinos de una solución son permutaciones de dos posiciones del vector. Como criterio de parada se tiene que no se mejore la solución actual o que se llegue a 100 iteraciones, o lo que es lo mismo, que la representación del árbol explorado llegue a una profundidad de 100.

Mejora de la eficiencia en las variantes del algoritmo genético

Al incorporar la búsqueda local al algoritmo genético, los tiempos se vieron afectados considerablemente, debido a ello en la búsqueda local, para no tener que calcular el fitness cada vez que se compruebe un nuevo vecino, se ha obtenido el fitness del nuevo vecino restándo al fitness anterior los valores antiguos relacionados con las dos posiciones cambiadas y sumando los valores nuevos relacionados con dichas posiciones. De esta forma se consiguió una notable mejora en los tiempos, lo cual pertmitió poder ejecutar con más generaciones y gracias a lo cual se vieron mejorados los resultados.

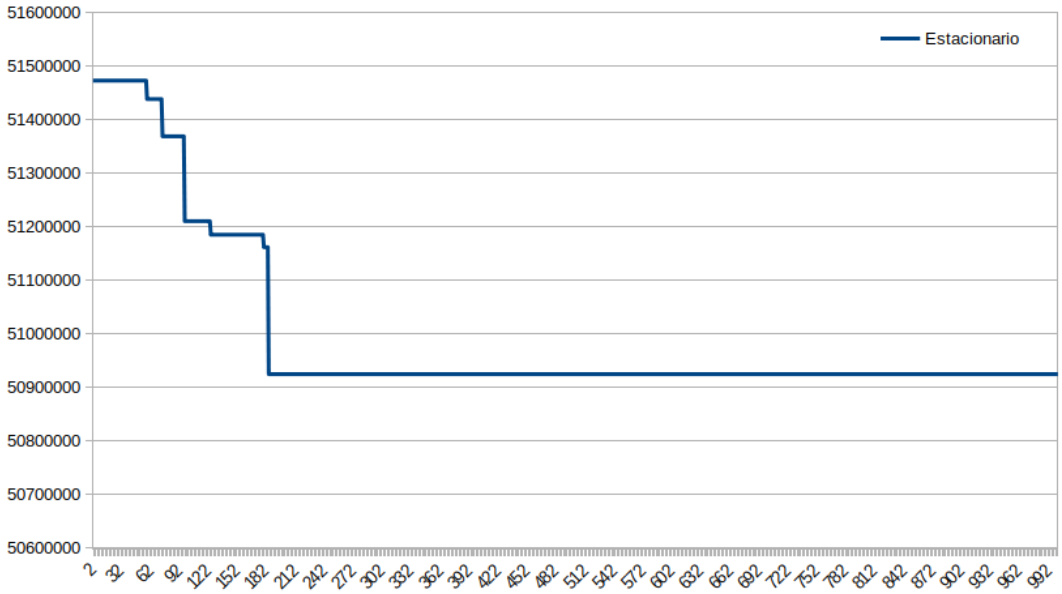
2. Experimentos realizados y resultados

Los algoritmos han sido implementados en C++, y ejecutados en un ordenador con procesador Intel Core i5, 16 GB de RAM y disco duro SSD en el sistema operativo Ubuntu 18.04 LTS.

2.1. Algoritmo genético estacionario

Aunque no se ha mencionado este algoritmo anteriormente, fue el primero por el que se optó. El algoritmo genético estacionario, que se implementó en un primer lugar, en cada generación solo seleccionaba 4 padres y los cruzaba con probabilidad 1, dando lugar a 2 hijos nuevos, uno por cada pareja de padres. Al solo haber 2 cruces por generación, el número de generaciones que se pueden ejecutar en un tiempo dado, es superior al del generacional, pero por otro lado tiene el inconveniente de que al solo haber 2 cruces, se estanca muy rápido y le cuesta salir de los óptimos locales a pesar de las mutaciones.

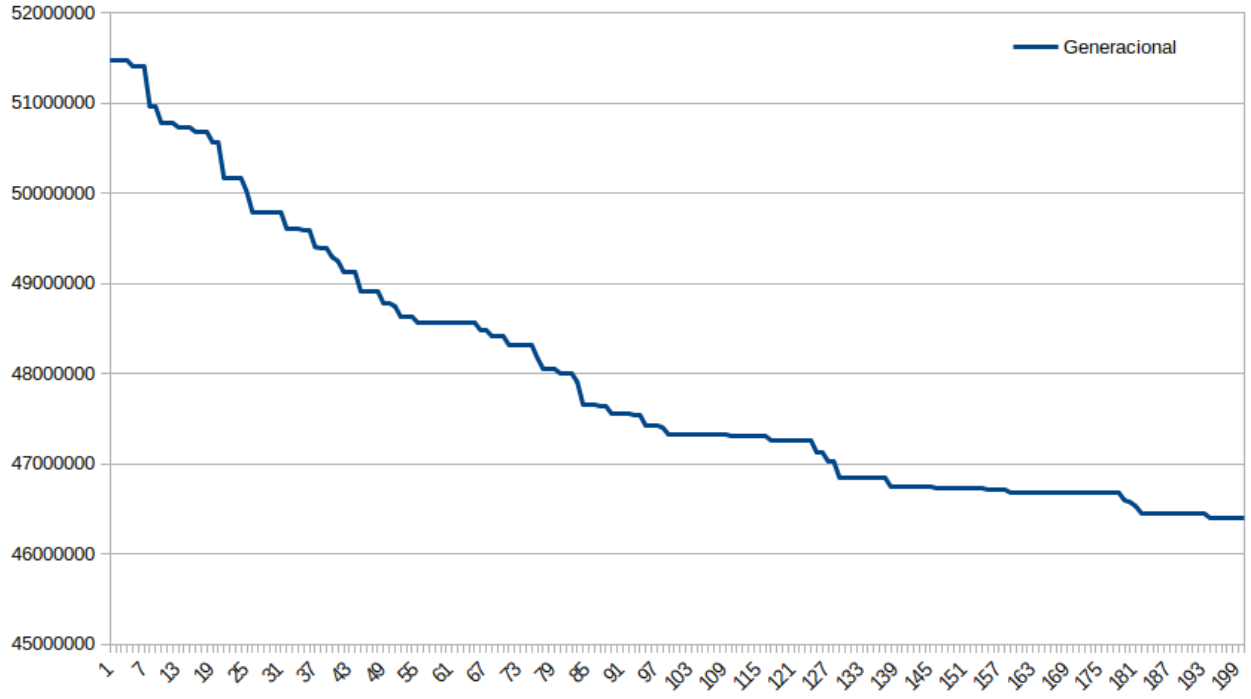
En la siguiente gráfica se muestra la evolución del valor de la función de coste, a lo largo de 1000 generaciones. El valor que se muestra de cada generación, se corresponde con el del mejor cromosoma de dicha generación.



2.2. Algoritmo genético generacional

Tras ver los resultados obtenidos por el algoritmo genético estacionario, pasamos a ver el comportamiento del generacional. Este algoritmo realiza muchos más cruces por generación, debido a lo cual el número de generaciones que podrán ejecutarse en un tiempo dado será inferior al del estacionario. Sin embargo veremos como hay una mejora mayor por cada generación en comparación con el algoritmo anterior, como resultado de un número elevado de cruces y la probabilidad de mutación que tiene cada descendiente.

En la siguiente gráfica se muestra la evolución del algoritmo generacional, donde el número de generaciones que van a ejecutarse para este experimento es de 200.



Podemos ver como en este caso hay una mejora constante, practicamente generación tras generación, especialmente al principio. Sin embargo esta mejora va decreciendo conforme van pasando las generaciones, ya estamos cada vez más cerca del óptimo y vemos como empieza a estancarse.

Finalmente se optó por elegir el algoritmo genético generacional para implementar las variantes baldwiniana y lamarckiana. Veremos a continuación una tabla comparativa de los 3 algoritmos.

Algoritmo	**Población**	**Generaciones**	**Coste**	**Tiempo**
Generacional	50	100	44956834	0.429631
Baldwiniano	50	100	44903700	69.671
Lamarckiano	50	100	44883614	67.138

Podemos ver como las variantes con la incorporación de la búsqueda local, han obtenido resultados mejores que el algoritmo genético básico. Sin embargo podemos ver como se ven afectados considerablemente los tiempos de ejecución.

El algoritmo que ha obtenido mejores resultados ha sido el lamarckiano. Debido a ello es el que vamos a seleccionar para intentar mejorar aún más el resultado. Para ello vamos a ver como mejora el coste al incrementar el número de generaciones con dicho algoritmo.

Algoritmo	**Población**	**Generaciones**	**Coste**	**Tiempo**
Lamarckiano	50	100	44883614	67.138
Lamarckiano	50	200	44883614	149.042
Lamarckiano	50	500	44859552	388.962
Lamarckiano	50	1000	44849384	798.97
Lamarckiano	50	2000	44802682	1593.574

Debido a que nos encontramos muy cerca del óptimo, se han tenido que incrementar bastante las generaciones poder ver alguna mejora en cada uno de los experimentos que se han mostrado en la tabla anterior. Los resultados han sido bastante satisfactorios, como podemos ver nos hemos quedado muy cerca del óptimo (44759294).

3. Manual de usuario

Para obtener el ejecutable a partir del código fuente, tan solo debemos situarnos en la carpeta donde se encuentra dicho código y escribir el comando make para generar el ejecutable.

El ejecutable podrá utilizarse del siguiente modo:

```
USO: ./main <fichero> <opcion>
Para elegir una opcion seleccione un numero:
    1: Algoritmo genetico estandar.
    2: Algoritmo genetico baldwiniano.
    3: Algoritmo genetico lamarckiano.
```