Metaheurísticas

Práctica 2 – Algoritmos genéticos

Grupo 3 – Martes 12:30-14:30

43180612K - Abad Vich, David
DAV00004@RED.UJAEN.ES
GRADO EN INGENIERÍA INFORMÁTICA Curso 2014/2015

Índice

1 CONTENIDOS DEL ARCHIVO

2 DESCRIPCIÓN DEL PROBLEMA

De forma general, se trata de encontrar una solución al problema de asignación cuadrática (del inglés, Quadratic Assignation Problem o QAP).

Para este problema, disponemos de *n* unidades y localizaciones, de modo que debemos asignar, de manera óptima, las unidades a las localizaciones. Para ello se dispone de un método para evaluar cómo de óptima es una solución y es mediante el flujo que hay entre diferentes unidades y la distancia que hay entre las localizaciones.

Sin embargo la descripción de encontrar una solución es bastante genérica. Debido a la cantidad de tiempo que nos puede llevar el conseguir la solución más óptima, dependiendo del tamaño del problema, en ocasiones nos tendremos que conformar con encontrar una solución, óptima por supuesto, pero no la más óptima que exista.

Entonces, el coste de una solución se puede formular como:

$$\underset{\pi \in \prod_N}{min} \left(\sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} \cdot d_{\pi(i)\pi(j))} \right)$$

donde:

- π es una permutación solución al problema, que representa la asignación de la unidad *i* a la localización π(i).
- f_{ij} es el flujo que hay de la unidad *i* a la *j*.
- d_{kl} es la distancia existente entre la localización *k* y la *l*.

Para este problema se usarán 20 casos diferentes, seleccionados de varios conjuntos de instancias disponibles en la QAPLIB (la biblioteca de problemas de asignación cuadrática) donde se puede encontrar además una descripción de lo que representa el problema.

3 APLICACIÓN DE LOS ALGORITMOS

Al aplicar los algoritmos al problema, hay que tener que tienen en común varios aspectos:

- Representación del problema: El problema se representa como dos matrices, una que contiene los flujos y otra las distancias, de manera que se puede ver el flujo que hay entre dos unidades y la distancia entre dos localizaciones de forma inmediata. Hay que tener en cuenta que mientras que la distancia entre dos localizaciones es la misma en ambos sentidos, puede que el flujo sea distinto entre dos unidades, dependiendo del sentido.

- Representación de las soluciones: Hemos mencionado antes la representación para la solución y es utilizando una permutación de *n* elementos (siendo *n* el tamaño del problema) a modo de que cada posición de la permutación representa una unidad y cada elemento de la permutación representa su localización asignada:

$$solución\text{:}\quad \begin{array}{c} unidad \\ localización \end{array} \begin{pmatrix} 1\ 2 \cdots n \\ i\ \ j\ \cdots k \end{pmatrix}\ i, j, k \in \{1, \dots, n\}$$

- Función objetivo: La función que se va a optimizar, es la suma del producto entre el flujo de dos unidades y la distancia de sus correspondientes localizaciones asignadas. Su pseudocódigo[1] es:

```
coste ← 0
tamaño ← tamaño(solución)
Para i ← 0 Hasta tamaño-1 Hacer
        Para j ← 0 Hasta tamaño-1 Hacer
                coste ← distancia[solución[i]][solución[j]] * flujo[i][j]
        Fin_Para
Fin_Para
```

- Generación de vecino: Para generar el entorno, se generarán los vecinos mediante el intercambio de dos elementos de la permutación solución.

- Función de factorización: Para un cálculo eficiente del coste de una solución, se extraerá del coste de la solución original los costes relacionados con las dos posiciones que se van a intercambiar para generar esa solución.

- Criterio de parada: Tras evaluar 10000 soluciones distintas, se detendrá la ejecución del algoritmo.

- Generación de la solución inicial aleatoria: En ambos algoritmos se utiliza la misma función para crear una solución inicial aleatoria de la que partir. El pseudocódigo es:

[1] Nota de programación: Hay que tener en cuenta que en el código los vectores y arrays empiezan desde la posición 0, por lo que queda representado en el pseudocódigo.

```
Función generarSolucionAleatoria() Devuelve un vector con la permutación
    tamaño ← tamaño del problema
    Para i ← 0 Hasta tamaño Hacer
        encontrada ← falso
        Mientras (no encontrada) Hacer
            crear localización aleatoria
            Para j ← 0 Hasta tamaño Hacer
                Si (localización está en el vector) Entonces
                    encontrada ← false
                Fin_Si
            Fin_Para
            vector[i] ← localización
        Fin_Mientras
    Fin_para
    Devolver vector
Fin_Función
```

- Mecanismo de selección considerado: El método para seleccionar a los padres consiste en un torneo binario, en el cual se seleccionan a dos padres para que compitan, obteniendo al final un solo ganador.

```
Función torneoBinario(padre1, padre2) Devuelve vector con la permutación del ganador
    Si (coste(padre1) < coste(padre2)) Entonces
        Devolver vector(padre1)
    Si_No
        Devolver vector(padre2)
    Fin_Si
Fin_Función
```

- Operadores de cruce: Operador de posición, que mantiene los elementos comunes de los padres y reordena el resto.

```
Función cruzaPadresPosición(padre1, padre2) Devuelve vector con el cruce
    tamaño ← tamaño(padre1)
    pos ← 0
    hijo ← llenar(-1)
    Para i ← 0 Hasta tamaño Hacer
        Si (padre1[i] == padre2[i]) Entonces
            hijo[i] ← padre1[i]
        Si_No
            no_asignados ← añade(padre1[i])
        Fin_Si
    Fin_Para
    Mientras (tamaño(no_asignados) > 0) Hacer
        Si (hijo[pos] == -1) Entonces
            gen ← extraer_aleatorio(no_asignados)
            hijo[pos] ← gen
        Fin_Si
        pos++
    Fin_Mientras
    Devolver hijo
Fin_Función
```

Operador de cruce OX, que toma una cadena central del primer padre y el resto de elementos, los toma siguiendo el orden del otro padre.

```
Función cruceOX(padre1, padre2) Devuelve vector con el cruce
        tamaño ← tamaño(padre1)
        hijo ← llenar(-1)
        inicio ← random(tamaño/2)
        fin ← inicio + tamaño/2
        Para i ← inicio Hasta fin Hacer
                hijo[i] ← padre1[i]
        Fin_Para
        Para i ← 0 Hasta tamaño Hacer
                está ← false
                Para j ← inicio Hasta fin Hacer
                        Si (padre2[i] == hijo[i]) Entonces
                                está ← true
                        Fin_Si
                        Si (No está) Entonces
                                elementos ← añadir(padre2[i])
                        Fin_Si
                Fin_Para
        Fin_Para
        Para i ← 0 Hasta tamaño Hacer
                Si (hijo[i] == -1) Entonces
                        hijo[i] ← saca primero(elementos)
                Fin_Si
        Fin_Para
        Devolver hijo
Fin_Funcion
```

4 ALGORITMOS UTILIZADOS

4.1 BÚSQUEDA LOCAL

Generamos una solución inicial aleatoria y exploramos el entorno, siguiendo la técnica del primer mejor, por lo que en cuanto encontramos una solución mejor, nos movemos a ella. El entorno se recorre

```
Procedimiento buscar()
        Generar solución inicial aleatoria
        tamaño ← tamaño(solución)
        Crear e inicializar array don't look bits dlb
        i ← 0
        mejora ← true
        Mientras (mejora AND soluciones_comprobadas < 10000) Hacer
                j ← i+1
                solución encontrada ← false
                Si (dlb[i] != 0) Entonces
                        //Si el elemento está desactivado, no entra en el siguiente bucle
                        encontrada ← true
                Fin_Si
                Mientras (j < tamaño AND no encontrada) Hacer
                        soluciones_comprobadas++
                        coste ← calcular el coste de cambiar las posiciones i y j
                        Si (coste < coste de la solución) Entonces
                                Se activa el elemento con dlb[i] ← 0
                                Se cambia la solución actual por la encontrada
                                encontrada ← true
                                i ← 0
                        Fin_Si
                        j++
                Fin_Mientras
                Si (no encontrada) Entonces
                        Se desactiva el elemento con dlb[i] ← 1
                Fin_Si
                Si (i == tamaño - 1) Entonces
                        //Si ya ha recorrido todos los elementos, es que no hay mejora
                        mejora ← false
                Fin_Si
                i++
        Fin_Mientras
Fin_Procedimiento
```

además de forma secuencial con la ayuda de la máscara *don't look bits*, por lo que se exploran sólo las unidades que están activas. Pseudocódigo:

En la búsqueda local, además de parar tras comprobar 10000 soluciones, también hay otro criterio de parada y es cuando no hay posibilidad de mejora en el entorno.

No habrá posibilidad de mejora cuando ya se han comprobado todos los cambios posibles, por lo que también la máscara se encontrará a 1 en todos sus bits (todos activados ya que no ha habido mejora para ninguno de los intercambios posibles).

4.2 AGG – Algoritmo Genético Generacional

Primero se realiza el cálculo de la cantidad de padres que se van a cruzar, según la probabilidad de cruce, y la cantidad de genes que se van a mutar, dependiendo de su respectiva probabilidad de mutación.

Se crea la generación 0 de forma aleatoria (con la función previamente descrita) y empieza el bucle.

En el bucle se siguen los pasos del algoritmo de: seleccionar, cruzar, mutar y reemplazar.

- **Selección**: Se realizan 50 torneos binarios entre padres escogidos aleatoriamente de la generación actual, que formarán la base de la nueva generación.

- **Cruce**: Se cruzan la cantidad esperada de padres, dos a dos, en orden, empezando desde el primero. El cruce realizado es el de posición o el OX, según se haya indicado para el algoritmo, generando dos hijos por cada cruce.

- **Mutación**: Se realizan la cantidad esperada de mutaciones, escogiendo aleatoriamente el cromosoma al que se le aplica y luego escogiendo dos posiciones aleatorias para el intercambio.

- **Reemplazamiento**: Se evalúa la nueva generación y antes de sustituir, se busca al mejor de la generación actual. Se sustituyen entonces los *n* primeros padres, comprobando que estamos quedándonos con el mejor.

Se repite hasta que se ha evaluado la función objetivo tantas veces como las indicadas (por defecto 20.000).

4.3 AGE – Algoritmo Genético Estacionario

Funciona de forma similar al AGG, teniendo en cuenta que pese a trabajar de forma parecida, éste incluye ciertos cambios que aquí comentaré.

Sigue el orden anterior de: seleccionar, cruzar, mutar y reemplazar.

- **Selección**: Se realizan dos torneos binarios aleatorios para seleccionar a los dos padres que se cruzarán.

- **Cruce**: Se cruzan los dos padres ganadores de los torneos. El cruce realizado es el de posición o el OX, según se haya indicado para el algoritmo, generando dos hijos por cada cruce.

- **Mutación**: Se realizan la cantidad esperada de mutaciones, escogiendo aleatoriamente el cromosoma al que se le aplica y luego escogiendo dos posiciones aleatorias para el intercambio.

- **Reemplazamiento**: Se evalúan los hijos y se buscan los dos peores de la generación actual. Si los hijos son mejores, entonces sustituyen a éstos.

Igual que en el anterior, se repite tantas veces como se le indique mediante la cantidad de evaluaciones.

5 ALGORITMO DE COMPARACIÓN – GREEDY

Este algoritmo está basado en la búsqueda del primer mejor. Consiste en realizar el sumatorio de cada fila de ambas matrices (flujos y distancias) y asignar a los mayores flujos, las menores distancias.

Es un algoritmo de bajo coste computacional, pues genera una solución (que variará según se asignen las localizaciones a las unidades) que normalmente no llegará a ser un óptimo, pero es probable que se encuentre cerca de uno, por lo que puede tener una gran aplicación si se utiliza para generar soluciones iniciales al problema, de las cuales explorar el entorno para encontrar una mejor solución.

```
Procedimiento CalcularSolucion()
        tamaño ← tamaño(distancias)
        Para i ← 0 Hasta tamaño-1 Hacer
                Para j ← 0 Hasta tamaño-1 Hacer
                        suma_distancias[i] ← suma_distancias[i] + distancias[i][j]
                        suma_flujos[i] ← suma_flujos[i] + flujos[i][j]
                Fin_Para
        Fin_Para
        Para i ← 0 Hasta tamaño-1 Hacer
                Para j ← 0 Hasta tamaño-1 Hacer
                        Si (suma_flujos[i] > max) Entonces
                                Coge nuevo flujo máximo
                        Fin_Si
                        Si (suma_distancias[i] < min AND suma_distancias[i] >= 0) Entonces
                                Coge nueva  distancia mínima
                        Fin_Si
                Fin_Para
                solución[flujo máximo] = distancia mínima
                suma_flujos[flujo máximo] = -1
                suma_distancias[distancia mínima] = -1
        Fin_Para
Fin  Procedimiento
```

Para generar la solución simplemente se suman los flujos y distancias y se asigna al mayor flujo, la menor distancia.

Para evitar volver a coger un elemento que ya ha sido asignado, se le asigna un valor negativo que pasará desapercibido cuando se busquen los siguientes valores máximo y mínimo.

6 PROCEDIMIENTO CONSIDERADO PARA EL DESARROLLO DE LA PRÁCTICA

Para el desarrollo de la práctica se ha utilizado código desarrollado por el propio alumno, con la ayuda de las transparencias de la asignatura y diferentes consultas al profesor.

Además, se han incorporado opciones extra para la práctica que pueden ayudar en el análisis de los resultados, tales como número de iteraciones o la posibilidad de usar la solución del algoritmo Greedy como solución inicial para ambas búsquedas.

6.1 PEQUEÑO MANUAL DE USUSARIO

La ejecución del programa es sencilla:

1. Ejecutar el archivo Meta1a.exe situado en la carpeta Deployment.
2. Elegir el archivo a ejecutar.
3. Introducir los diferentes parámetros (opcional, se puede dejar por defecto):
    a. Matriz que se lee primero (por defecto la de flujos).
    b. Valor de la semilla para la generación de números aleatoria (por defecto 123456).
    c. (Opcional) Elegir el número de evaluaciones de la función objetivo a realizar para los algoritmos genéticos.
4. Hacer clic en ejecutar.

El programa incluye una pequeña barra de progreso, introducido de forma manual, para mostrar que la ejecución sigue activa.

También cabe la posibilidad de ejecutar el programa usando Qt, para ello, una vez instalado Qt, hay que abrir el proyecto que desde la pantalla de bienvenida de Qt, es haciendo clic en la opción Open Project, seleccionando la carpeta donde se encuentra el proyecto y seleccionando el archivo .pro que hay ahí.

Una vez abierto, construir y ejecutar. Saldrá entonces la misma interfaz que con el ejecutable y ya hay que seguir los mismos pasos que con el mismo.

NOTA: Es posible que con archivos de gran tamaño (como lipa90) el programa parece no responder, pero acaba la ejecución transcurridos unos segundos.

7 EXPERIMENTOS Y ANÁLISIS DE RESULTADOS

Para todos los problemas se ha considerado siempre el mismo valor para la semilla: 123456.

Pasamos a comentar las instancias usadas en esta práctica[2], todas partes de la QAPLIB:

- bur*: Tenemos una matriz que contiene el tiempo medio de escritura de una máquina de escribir y otra con la frecuencia de los pares de letras en diferentes idiomas.
- chr*: Nos dan una matriz de adyacencia de un árbol balanceado y otra matriz con grafos completos.
- els19: Las distancias entre 19 instalaciones diferentes de un hospital y el flujo de pacientes entre ellas. Hay que notar que ésta es de las instancias que tendrá que leerse primero la matriz de distancias.
- esc32a: Ejemplos de una aplicación, usados para circuitos secuenciales auto-comprobables.
- kra32: Contienen datos reales usados para planear el Hospital Regensburg en Alemania.
- lipa90a: Esta instancia proviene de generadores de problemas, que proveen de instancias asimétricas con soluciones óptimas conocidas.
- nug25: La matriz de distancias contiene las distancias de cuadrículas rectangulares de Manhattan. También hay que leer primero la matriz de distancias en esta instancia.
- sko*: Las distancias de éstos problemas son rectangulares y los flujos son número pseudoaleatorios. Otra más para la que se lee primero la matriz de distancias.
- tai*: Instancias uniformemente generadas. Asimétricas y aleatorias, suceden en la generación de patrones de grises.
- tho40: Las distancias de este problema son rectangulares, por lo que la primera matriz que se lee es la de distancias.

[2] En caso de no indicar nada, la matriz que se ha de leer primero es la de flujos.

7.1 RESULTADOS GREEDY

Algoritmo Greedy			
Caso	**Coste obtenido**	**Desv**	**Tiempo**
Els19	38627698	124,42	0,00
Chr20a	8632	293,80	0,00
Chr25a	17556	362,49	0,00
Nug25	4462	19,18	0,00
Bur26a	5968117	9,98	0,00
Bur26b	4273373	11,93	0,00
Tai30a	2098700	15,43	0,00
Tai30b	1387185541	117,73	0,00
Esc32a	342	163,08	0,00
Kra32	117130	32,05	0,00
Tai35a	2952758	21,91	0,00
Tai35b	506552163	78,79	0,00
Tho40	312578	29,96	0,00
Tai40a	3730054	18,82	0,00
Sko42	18902	19,54	0,00
Sko49	27378	17,07	0,00
Tai50a	5804880	17,54	1,00
Tai50b	788404422	71,83	0,00
Tai60a	8345630	15,82	1,00
Lipa90a	367797	1,99	1,00

7.2 Resultados Búsqueda Local

Algoritmo BL			
Caso	**Coste obtenido**	**Desv**	**Tiempo**
Els19	23734354	37,89	1,00
Chr20a	3912	78,47	0,00
Chr25a	8028	111,49	2,00
Nug25	3962	5,82	1,00
Bur26a	5444522	0,33	2,00
Bur26b	3851727	0,89	2,00
Tai30a	1916134	5,39	2,00
Tai30b	691972163	8,61	3,00
Esc32a	170	30,77	3,00
Kra32	98630	11,20	5,00
Tai35a	2585098	6,73	5,00
Tai35b	353867912	24,90	5,00
Tho40	264608	10,02	9,00
Tai40a	3339928	6,39	8,00
Sko42	16234	2,67	11,00
Sko49	24972	6,78	22,00
Tai50a	5265454	6,61	19,00
Tai50b	487830146	6,32	34,00
Tai60a	7717196	7,09	37,00
Lipa90a	363582	0,82	227,00

7.3 Resultados AGG – Posición

Algoritmo AGG - Posición			
Caso	**Coste obtenido**	**Desv**	**Tiempo**
Els19	28194616	63,80	77,00
Chr20a	5586	154,84	82,00
Chr25a	10906	187,30	95,00
Nug25	4326	15,54	94,00
Bur26a	5570802	2,66	99,00
Bur26b	3878353	1,58	96,00
Tai30a	2066634	13,67	110,00
Tai30b	872032694	36,87	109,00
Esc32a	294	126,15	116,00
Kra32	119630	34,87	118,00
Tai35a	2734538	12,90	132,00
Tai35b	397844320	40,42	137,00
Tho40	304106	26,44	153,00
Tai40a	3541596	12,81	151,00
Sko42	18776	18,75	156,00
Sko49	27060	15,71	181,00
Tai50a	5668950	14,78	186,00
Tai50b	633037887	37,97	184,00
Tai60a	8252582	14,52	277,00
Lipa90a	366454	1,61	394,00

7.4 RESULTADOS AGG - OX

Algoritmo AGG - OX			
Caso	**Coste obtenido**	**Desv**	**Tiempo**
Els19	17374134	0,94	67,00
Chr20a	2986	36,22	70,00
Chr25a	6444	69,76	84,00
Nug25	3912	4,49	83,00
Bur26a	5443710	0,31	86,00
Bur26b	3836094	0,48	87,00
Tai30a	1933380	6,34	99,00
Tai30b	712959148	11,90	101,00
Esc32a	160	23,08	107,00
Kra32	96070	8,31	106,00
Tai35a	2591310	6,99	122,00
Tai35b	328216981	15,85	126,00
Tho40	285844	18,85	141,00
Tai40a	3491958	11,23	144,00
Sko42	17158	8,51	146,00
Sko49	25624	9,57	174,00
Tai50a	5613688	13,67	180,00
Tai50b	486249843	5,98	178,00
Tai60a	8163444	13,29	226,00
Lipa90a	366243	1,56	414,00

7.5 Resultados AGE – Posición

Algoritmo AGE - Posición			
Caso	**Coste obtenido**	**Desv**	**Tiempo**
Els19	29363872	70,60	32,00
Chr20a	6770	208,85	32,00
Chr25a	13006	242,62	41,00
Nug25	4564	21,90	40,00
Bur26a	5525618	1,82	43,00
Bur26b	3973631	4,08	39,00
Tai30a	2095042	15,23	47,00
Tai30b	916472722	43,85	47,00
Esc32a	274	110,77	49,00
Kra32	123070	38,75	49,00
Tai35a	2812434	16,12	60,00
Tai35b	393450872	38,87	70,00
Tho40	310744	29,20	71,00
Tai40a	3644410	16,09	75,00
Sko42	18982	20,05	74,00
Sko49	27658	18,27	91,00
Tai50a	5697150	15,36	95,00
Tai50b	649542376	41,57	95,00
Tai60a	7555590	4,85	130,00
Lipa90a	363939	0,92	252,00

7.6 Resultados AGE – OX

Algoritmo AGE - OX			
Caso	**Coste obtenido**	**Desv**	**Tiempo**
Els19	26780374	55,59	53,00
Chr20a	5910	169,62	56,00
Chr25a	12274	223,34	65,00
Nug25	4230	12,98	68,00
Bur26a	5558831	2,44	67,00
Bur26b	3883095	1,71	67,00
Tai30a	2066650	13,67	79,00
Tai30b	787894522	23,67	78,00
Esc32a	286	120,00	82,00
Kra32	115250	29,93	82,00
Tai35a	2799328	15,58	98,00
Tai35b	346698313	22,37	107,00
Tho40	308088	28,09	114,00
Tai40a	3622450	15,39	119,00
Sko42	17936	13,43	119,00
Sko49	25868	10,61	155,00
Tai50a	5665576	14,72	150,00
Tai50b	623303870	35,85	148,00
Tai60a	7604606	5,53	195,00
Lipa90a	363893	0,90	389,00

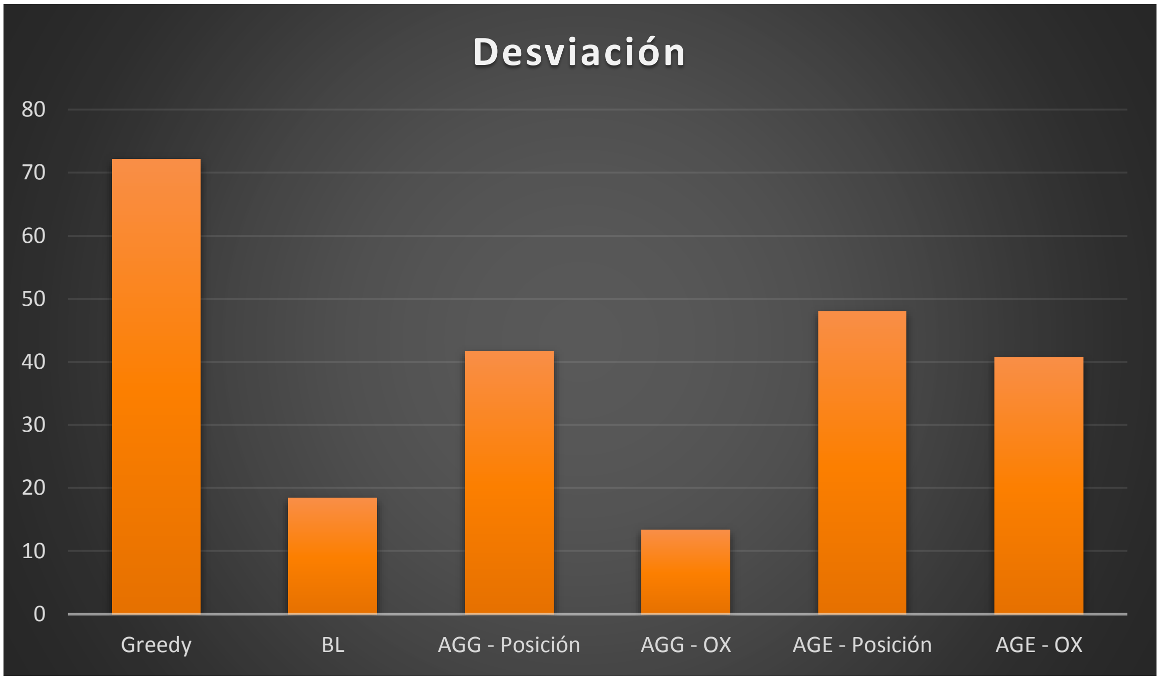
7.7 Resultados Globales

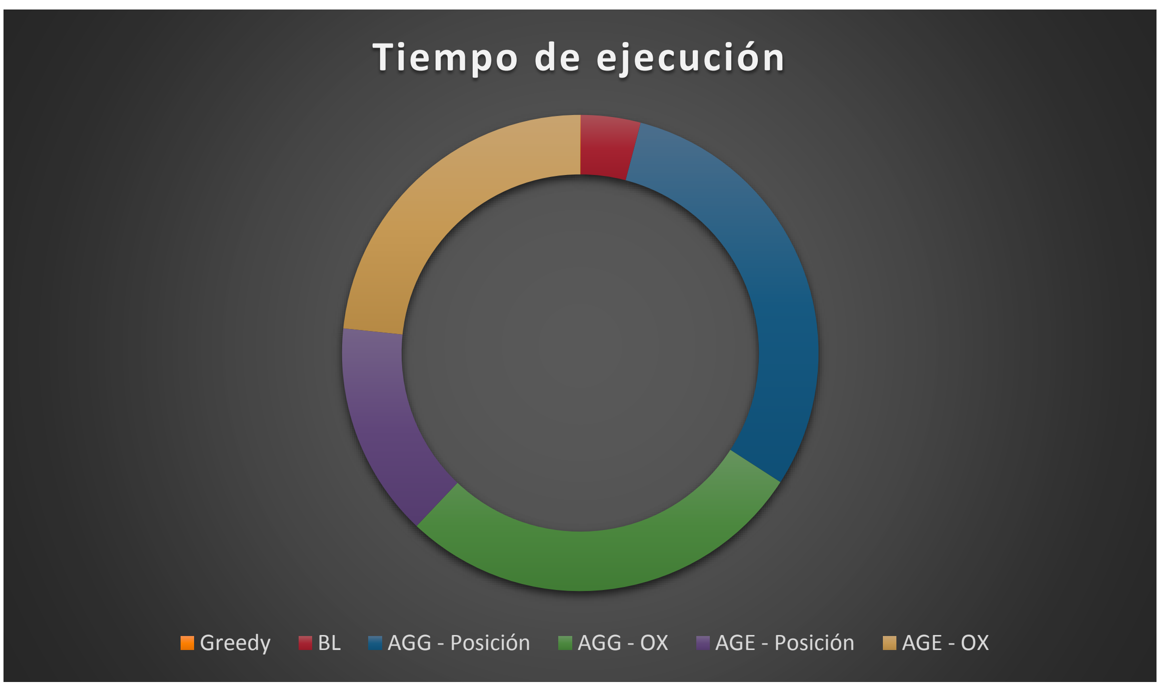
Algoritmo	*Desviación*	*Tiempo (en ms.)*
Greedy	72.17	0.15
BL	18.46	19.90
AGG - Posición	41.66	147.35
AGG - OX	13.37	137.05
AGE - Posición	47.99	71.60
AGE - OX	40.77	114.55

A excepción del generacional con el operador de cruce OX, el resto de genéticos no han obtenido muy buenos resultados. Aunque los genéticos suelen ser buenos resolviendo problemas que son representables mediante una permutación, cuando son sobre distancias producen mejores resultados, por lo que en el caso del QAP no es buena idea aplicar genéticos.

Realizar cambios en el tamaño de la población produce diversos resultados, en ciertos problemas nos proporciona una pequeña mejora (problemas como los bur* o chr*, aumentando y disminuyendo el tamaño en 10 respectivamente), sin embargo para hacer un estudio general no da una buena base, ya que hay que cambiarlo según el problema.

En el caso de la cantidad de evaluaciones sí que se ve una mayor mejora conforme se incrementa la cantidad de las mismas. Aunque esta mejora no es directamente proporcional, si no que al ir aumentando, cada vez se precisa de una mayor cantidad de evaluaciones para obtener otra mejora considerable.


Desviación
80
70
60
50
40
30
20
10
0
Greedy
BL
AGG - Posición
AGG - OX
AGE - Posición
AGE - OX


Tiempo de ejecución
Greedy
BL
AGG - Posición
AGG - OX
AGE - Posición
AGE - OX

8 BIBLIOGRAFÍA

- G. Ochoa, S. Verel y M. Tomassini. First-improvement vs. Best-improvement Local Optima Networks of NK Landscapes. *11th International Conference on Parallel Problem Solving From Nature*, Krakow, Poland, 2010.

- Loiola. A survey for the quadratic assignment problem. 2007