Práctica 5

Algoritmos Genéticos para el
Problema de la Asignación Cuadrática
AM−(10,1.0) • AM−(10,0.1) • AM−(10,0.1mej)

Mario Ruiz Calvo
mariorc@correo.ugr.es
33958755-Z

Grupo 2
(J 17:30-19:30)

Curso 2014-2015

INDICE

1. Descripción del Problema

El problema de asignación cuadrática es un problema de optimización combinatioria.. En éste se trata de asignar N unidades a una cantidad N de localizaciones en donde se considera un costo asociado a cada una de las asignaciones. Este costo dependerá de las distancias y flujo entre las instalaciones. Por lo que el problema consiste en encontrar la asignación óptima de caada unidad a una localización. La función objetivo mide la bondad de una asignación, considerada el producto de la distancia entre cada par de localizaciones y el flujo que circula entre cada par de unidades.

2. Componentes de los algoritmos

- Esquema de representación: La solución se representa con un vector en forma de permutación π de tamaño n, donde los indices del vector corresponden a las unidades y el contenido a las distancias.

- Función objetivo:

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

- Generación de soluciones aleatorias:

```
GenerarSolucionAleatoria(sol)
        Desde i=1 hasta n (tamaño del problema)
                Hacer
                        r = rand(0,n)
                Mientras r este contenido en sol
                sol[i] = r
        Fin
Fin
```

- Descripción del algoritmo de búsqueda local:

```
BúsquedaLocalPrimeroMejor(sol, coste,nevaluaciones)
        Hacer
        n_vecinos=0
                        Hacer
                                espacioCompleto=generarVecino(sol, svec, r, s, reinciar)
                                reiniciar=false
                                costoFactorizado(sol, r, s, coste_vecino)
                                Nevaluaciones = nevaluaciones +1
                                nvecinos=nvecions+1
                        Mientras (coste_vecino>=0 && !espacioCompleto && nvecinos<400)

                        Actualizar MejorSolucion (sol, svec, coste, coste_vecino)
                        Si coste_vecino<0 reinciar=true

        Mientras (coste_vecino<0 &&)
Fin
```

- Coste factorizado:

$$\sum_{k=1,k\neq r,s}^{n} \begin{bmatrix} f_{rk}\cdot\left(d_{\pi(s)\pi(k)}-d_{\pi(r)\pi(k)}\right)+f_{sk}\cdot\left(d_{\pi(r)\pi(k)}-d_{\pi(s)\pi(k)}\right) \\ f_{kr}\cdot\left(d_{\pi(k)\pi(s)}-d_{\pi(k)\pi(r)}\right)+f_{ks}\cdot\left(d_{\pi(k)\pi(r)}-d_{\pi(k)\pi(s)}\right) \end{bmatrix}$$

- Generar vecinos para la búsqueda local: El tamaño del entorno es $\frac{n\cdot(n-1)}{2}$

```
                        generarVecino(sol,vecino, r, s, reiniciar)

                                Static i=0,j=1

                                Si reiniciar
                                                I=0, =01
                                Si i==n-1
                                                Devolver true

                                r=i, s=j, vecino=sol

                                Intercambiar (vecino[i], vecino[j]

                                Si j>n-1
                                                i=i+1
                                                j=i+1
                                Sino
                                                j=j+1

                                Devolver false
                        Fin
```

- Descripción del mecanismo de selección:

```
seleccion(poblacion,padres,npadres)
        Desde i=1 hasta npadres hacer
                        j=rand(0,N_CROMOSOMAS)
                        k=rand(0,N_CROMOSOMAS) (k distinto de j)
                        Si ( coste(poblacion[j]) < coste(poblacion[k]) )
                                padres[i]=poblacion[j]
                        Sino
                                padres[i]=poblacion[k]
        Fin
Fin
```

- Descripción del operador de cruce:

```
crucePosicion(padres,ncruces)
        Desde i=1 hasta ncruces hacer
                        Desde j=1 hasta n hacer
                                Si padres[i*2][j] != padres[(i*2)+1][j]
                                        Insertar j en posiciones
                                        Insertar padres]i*2][j] en asignaciones
                                FinSI
                        Fin
                        Mientras posiciones no sea vacio
                                pos=rand(0,posiciones.size())
                                padres[i*2][posiciones[pos]]=asignaciones[0]
                                Eliminar posiciones[pos]
                                Eliminar asignaciones[0]
                        FinMientras
        Fin
Fin
```

- Descripción del operador de mutación:

```
mutacion(padres,npadres,prob_mutacion)
        Desde i=1 hasta npadres
                        p=rand(0,1)
                        Si prob_mutacion<p
                                GeneraVecinoAleatorio(padres[i])
        Fin
Fin
```

```
GeneraVecinoAleatorio(sol)
        r=rand(0,n)
                s=rand(0,n) (r distinto de s)
                intercambiar(sol[r],sol[s])
Fin
```

Para evitar genererar muchos números aleatorios que después no se van a utilizar en el esquema generacional, se calcula previamente el número de mutaciones que se van a realizar como *prob_mutacion*n*N_CROMOSOMAS* y se mutan directamente.

```
mutacionG(padres,npadres,nmutaciones)
        Desde i=1 hasta nmutaciones
                p=rand(0,N_CROMOSOMAS) (p no repetido)
                GeneraVecinoAleatorio(padres[p])
        Fin
Fin
```

3. Algoritmos Meméticos

La Búsqueda Local que se aplica a la población de cromosomas se lleva a acabo en la función reeemplazar, siendo necesarias 3 funciones distintas según el tipo de algoritmo memético.

```
reemplazamientoT(poblacion,padres,mejor_sol,n_evaluaciones,aplicar_local)
        poblacion=padres
        Si Min(costes(padres)) == coste(mejor_sol)
                poblacion[Max(costes(padres))]=coste(mejor_sol)
        Si aplicar_local
                Desde i=1 hasta N_CROMOSOMAS
                        BusquedaLocalPrimeroMejor(poblacion[i],costes[i],n_evaluaciones)
Fin
```

```
reemplazamientoS(poblacion,padres,mejor_sol,n_evaluaciones,aplicar_local,pls)
        poblacion=padres
        Si Min(costes(padres)) == coste(mejor_sol)
                poblacion[Max(costes(padres))]=coste(mejor_sol)
        Si aplicar_local
                Desde i=1 hasta N_CROMOSOMAS
                        Si rand()<pls
                                BusquedaLocalPrimeroMejor(poblacion[i],costes[i],n_evaluaciones)
Fin
```

```
reemplazamientoM(poblacion,padres,mejor_sol,n_evaluaciones,aplicar_local)
        poblacion=padres
        Si Min(costes(padres)) == coste(mejor_sol)
                poblacion[Max(costes(padres))]=coste(mejor_sol)
        Si aplicar_local
                ordenar(poblacion) (de mayor a menor)
                Desde i=1 hasta N_CROMOSOMAS*0,1
                        BusquedaLocalPrimeroMejor(poblacion[i],costes[i],n_evaluaciones)
Fin
```

Se utilizará el mismo esquema para las tres versiones del algoritmo memético cambiando solo la función reemplazamiento utilizando la propia de cada problema.

```
esquemaGenerico(sol,coste, prob_cruce, prob_mutacion)
        Ncruces = prob_cruce*(N_CROMOSOMAS)/2
        Nmutaciones = prob_mutacion*n*N_CROMOSOMAS
        aplicar_local = false

        inicializar(poblacion,padres)
        evaluar(poblacion, costes_poblacion, sol, coste)

        Desde i=1 hasta 25000 (i+=N_CROMOMSOMAS)
                seleccion(poblacion,padres,npadres)
                cruce(padres,ncruces)
                mutacion(padres,npadres,prob_mutacion)
                aplicar_local = (ngeneraciones==0 && i!=0)
                reemplazamientoX(poblacion,padres,mejor_sol, i, aplicar_local [, p])
                evaluar(poblacion, costes_poblacion, sol_l, coste_l)
                ActualizarMejorSolucion(sol, sol_l,coste,coste_l)
        FinDesde
Fin
```

4. Procedimiento para la práctica

Para el desarrollo de la práctia se ha partido de un código desde cero, siguiendo como referencia el código proporcionado en la plataforma DECSAI y en el pseudocódigo y explicación de los seminarios y las tranparencias de teoría.

La practica se orgainza en las siguientes carpetas:

- bin: contiene los ficheros ejecutables
- data: contiene los ficheros .dat de donde se extraen los flujos y distancias.
- Include: contiene las ficheros .h
- lib: contien la librerias con todos lof ficheros objeto
- obj: contiene los ficheros objeto .o
- resultados: contiene las tablas con los resultados que se han obtenido para los distintos algoritmos y las desviaciones de tiempo y costo.
- Src: contiene el código fuente.

Además el directorio raíz contiene un fichero makefile para compilar los ejecutables y dos scripts con los que se obtienen los resultados en un formato fácil de trasladar a las tablas.

Ejecutando el fichero makefile se obtienen dos ejecutables: main y obtener_resultados:

main: Ejecuta la búsqueda local primer mejor, la búsqueda multiarranque básica y la busqueda local retroactiva, con una semilla fija (5555) , mostrando el costo obtenido y el tiempo.

-modo de empleo (desde directorio practica2):

./bin/main data/nombre_del_fichero.dat

obtener_resultado: Realiza una ejecución de un algoritmo pasado por parámetro con una semilla variable (también pasada como parámetro) e imprime el coste y el tiempo de ejcución en un formato adecuado para la extracción de datos para las 20 instancias.

-modo de empleo (desde directorio practica2):

./bin/obtener_resultado id_algoritmo semilla

identificador de algoritmo:
- 0: greedy
- 1: AM-(10,1.0)
- 2: AM-(10,0.1)
- 3: AM-(10,0.1mej)

Ejecutando el script obtener_resultado2.ssh se realizan una ejecuciones DE LAS 20 INSTANCIAS del algoritmo pasado por parámetro. Se imprimran los resultados en un formato adecuado para recoger los datos en tablas.

-modo de empleo (desde directorio practica2):

./obtener_resultados2.ssh id semilla > resultados/nombre

5. Análisis de resultados

Para obtener los resulados se ha realizado una única ejecucion de las 20 instancias con semilla 5555.

Greedy					
Caso	**Desv**	**Tiempo**	**Caso**	**Desv**	**Tiempo**
Chr20b	342,82	0,000046	**Sko64**	18,39	0,000132
Chr20c	456,63	0,00003	**Sko72**	15,16	0,000161
Chr22a	130,31	0,000045	**Sko81**	15,11	0,000208
Chr22b	134,48	0,000034	**Sko90**	13,49	0,000278
Els19	124,42	0,000027	**Sko100a**	14,37	0,000185
Esc32b	140,48	0,00007	**Sko100b**	13	0,000207
Kra30b	28,04	0,000056	**Sko100c**	12,89	0,00018
Lipa90b	29,36	0,000422	**Sko100d**	14,07	0,000227
Nug30	21,69	0,00002	**Sko100e**	14,64	0,000178
Sko56	16,09	0,000104	**Wil50**	10,59	0,000056

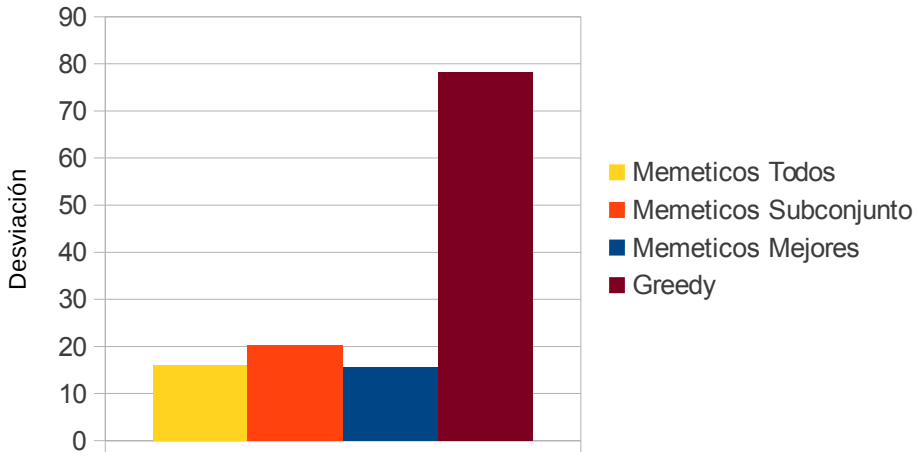
Memeticos Todos					
Caso	**Desv**	**Tiempo**	**Caso**	**Desv**	**Tiempo**
Chr20b	**45,95**	0,451981	Sko64	12,3	3,383867
Chr20c	**83,31**	0,461414	Sko72	10,37	4,312914
Chr22a	**7,05**	0,590017	Sko81	**11,54**	4,653349
Chr22b	**10,04**	0,588104	Sko90	12,27	5,480952
Els19	30,1	0,414146	Sko100a	**3,96**	7,151489
Esc32b	**28,57**	1,480248	Sko100b	4,14	6,779602
Kra30b	**4,21**	1,40774	Sko100c	5,66	7,111238
Lipa90b	25,41	4,999515	Sko100d	**3,73**	7,019949
Nug30	5,94	1,387815	Sko100e	4,89	6,890097
Sko56	**5,26**	2,897747	Wil50	4,83	2,651749

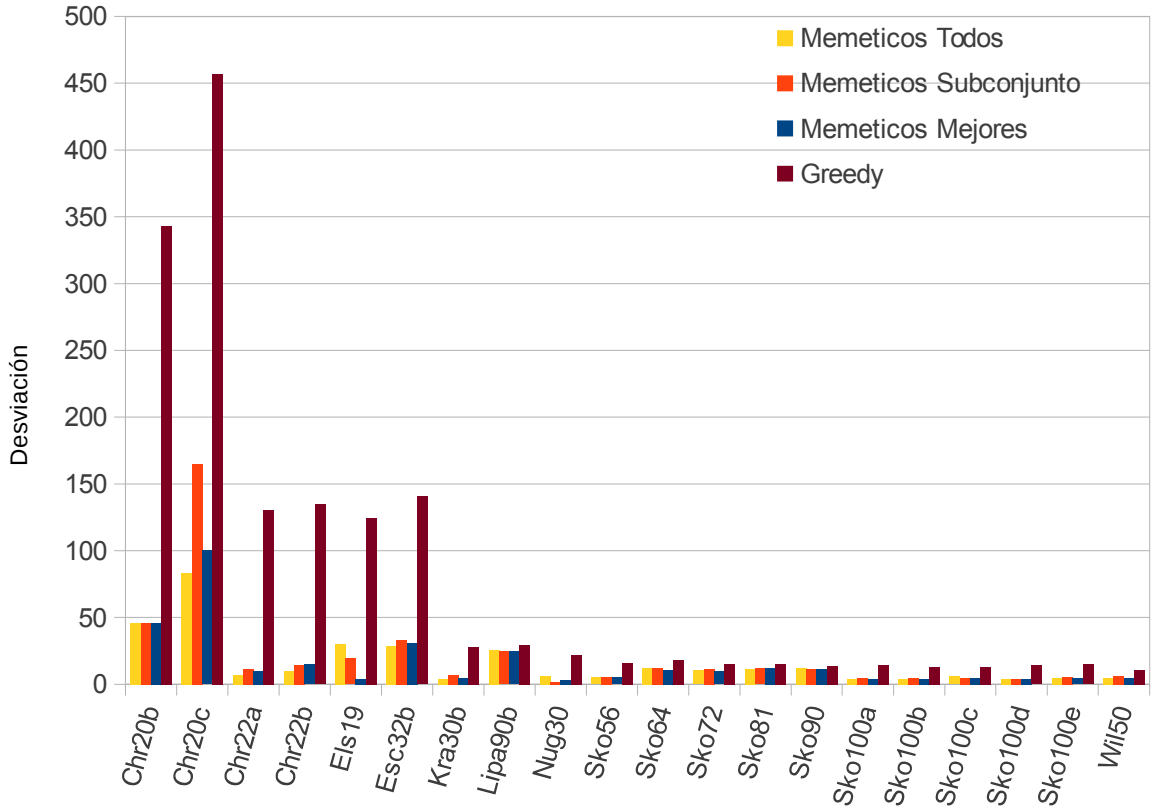
Memeticos Subconjunto					
Caso	**Desv**	**Tiempo**	**Caso**	**Desv**	**Tiempo**
Chr20b	**45,95**	0,159951	Sko64	11,83	1,062057
Chr20c	165,05	0,120004	Sko72	11,46	1,301353
Chr22a	11,5	0,148802	Sko81	11,71	1,607994
Chr22b	14,43	0,149922	Sko90	**11,24**	1,844921
Els19	19,31	0,105126	Sko100a	4,64	2,528875
Esc32b	33,33	0,320014	Sko100b	4,56	2,333826
Kra30b	6,96	0,340383	Sko100c	5	2,562811
Lipa90b	**24,51**	1,925832	Sko100d	**3,73**	2,655841
Nug30	**1,89**	0,326999	Sko100e	5,61	2,552207
Sko56	5,58	0,868758	Wil50	6,26	0,783241

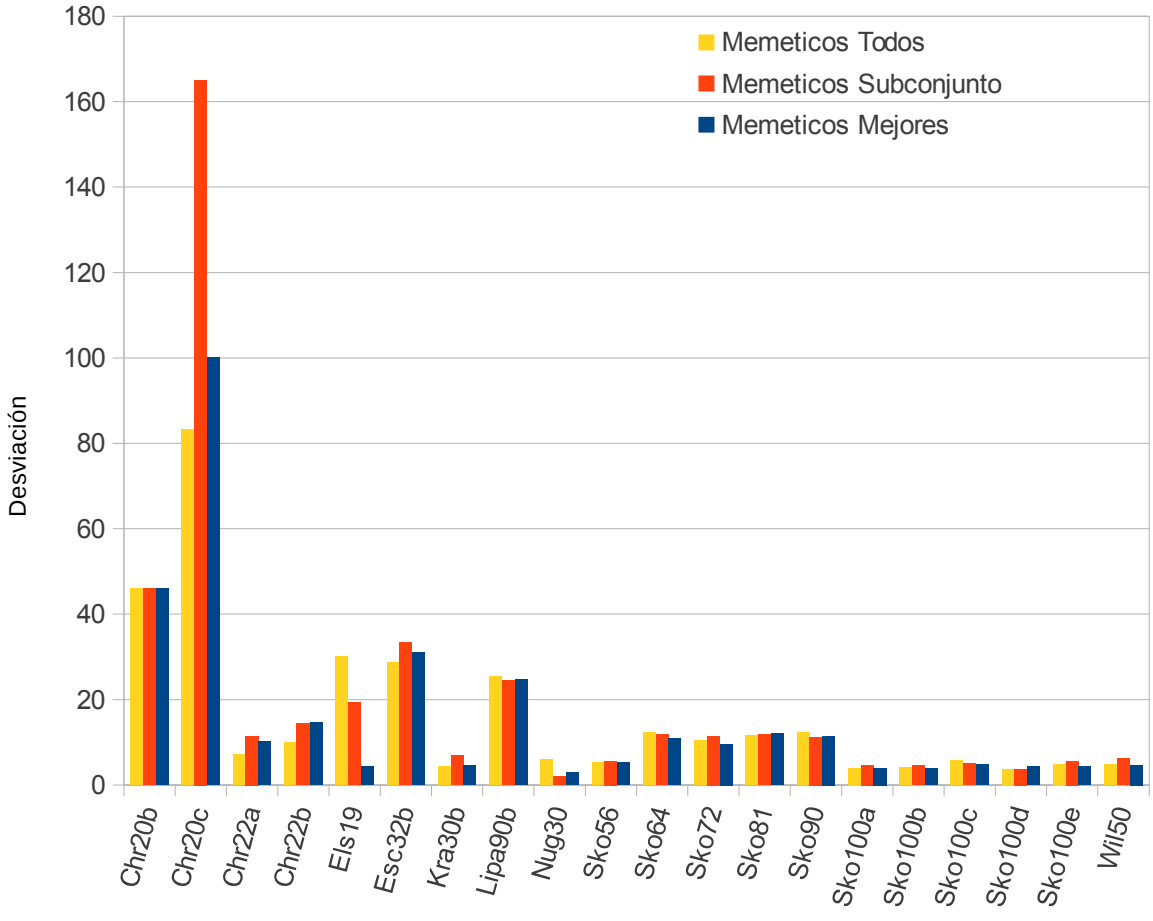
Caso	Desv	Tiempo
Chr20b	**45,95**	0,12195
Chr20c	100,16	0,119152
Chr22a	10,17	0,146021
Chr22b	14,66	0,146774
Els19	**4,21**	0,10901
Esc32b	30,95	0,325354
Kra30b	4,51	0,300126
Lipa90b	24,61	1,855774
Nug30	2,91	0,310486
Sko56	**5,26**	0,83013

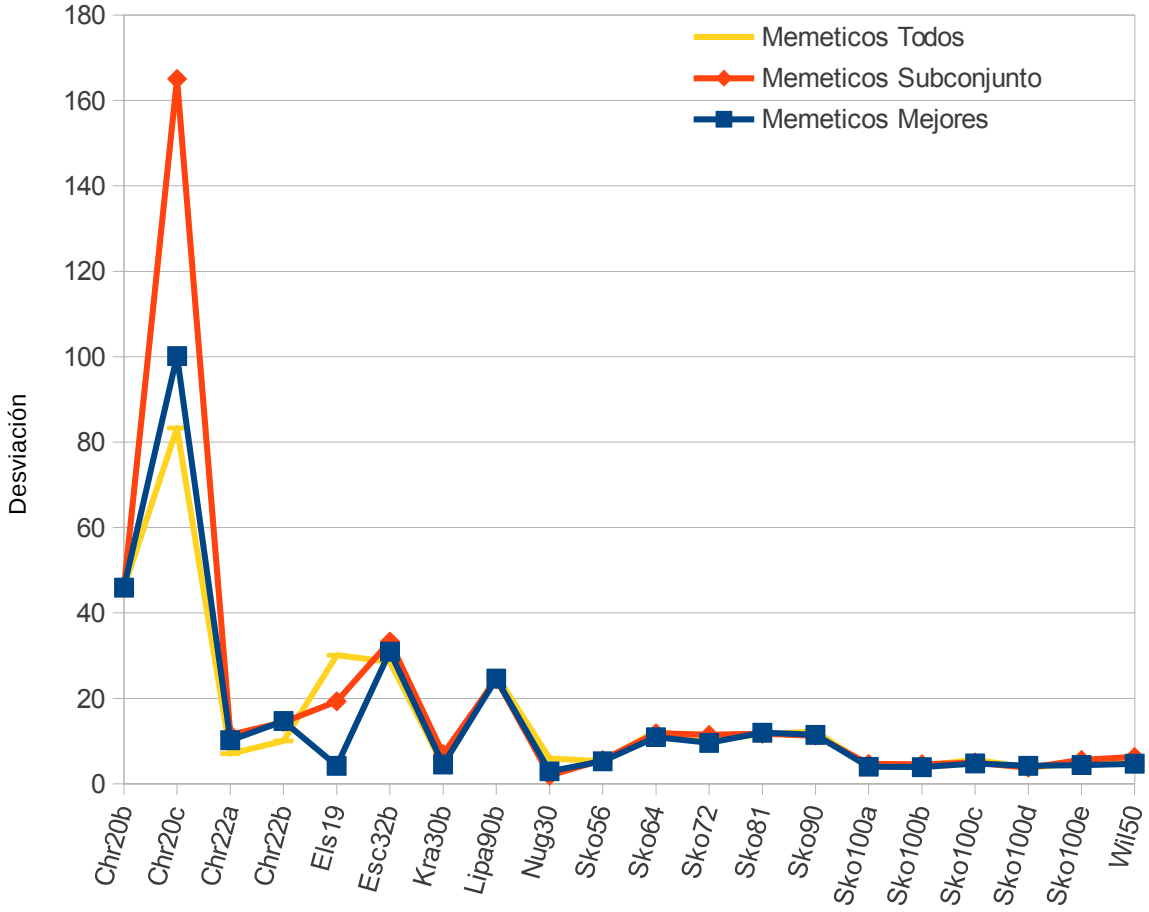
Caso	Desv	Tiempo
Sko64	**10,92**	1,047585
Sko72	**9,59**	1,283406
Sko81	11,96	1,546033
Sko90	11,46	1,868233
Sko100a	3,97	2,446804
Sko100b	**3,91**	2,382026
Sko100c	**4,73**	2,319614
Sko100d	4,21	2,402884
Sko100e	**4,36**	2,383014
Wil50	**4,68**	0,68413

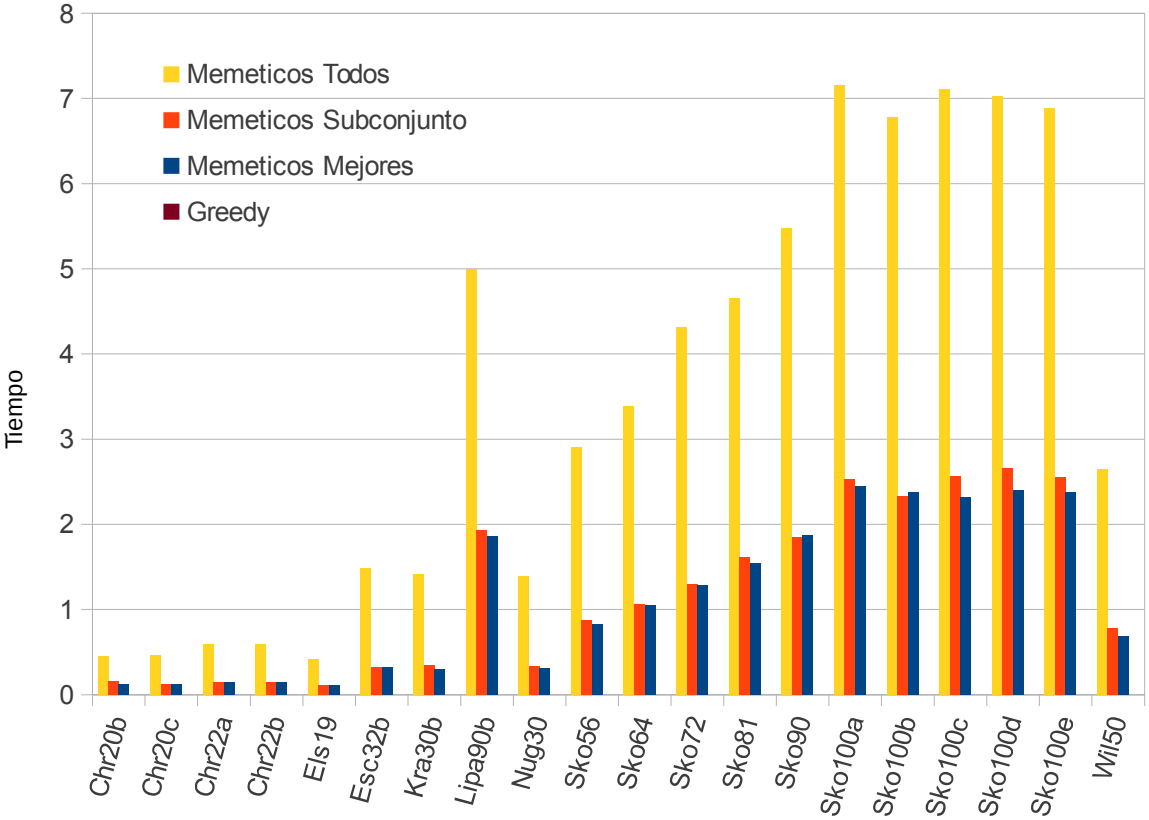
Algoritmo	Desv	Tiempo
Memeticos Todos	15,98	3,50569665
Memeticos Subconjunto	20,23	1,18494585
Memeticos Mejores	**15,66**	1,13
Greedy	78,3	0,0001333




Desviación
500
450
400
350
300
250
200
150
100
50
0
Memeticos Todos
Memeticos Subconjunto
Memeticos Mejores
Greedy
Chr20b
Chr20c
Chr22a
Chr22b
Els19
Esc32b
Kra30b
Lipa90b
Nug30
Sko56
Sko64
Sko72
Sko81
Sko90
Sko100a
Sko100b
Sko100c
Sko100d
Sko100e
Wil50


Desviación
180
160
140
120
100
80
60
40
20
0
Memeticos Todos
Memeticos Subconjunto
Memeticos Mejores
Chr20b
Chr20c
Chr22a
Chr22b
Els19
Esc32b
Kra30b
Lipa90b
Nug30
Sko56
Sko64
Sko72
Sko81
Sko90
Sko100a
Sko100b
Sko100c
Sko100d
Sko100e
Wil50


Memeticos Todos
Memeticos Subconjunto
Memeticos Mejores
Desviación
180
160
140
120
100
80
60
40
20
0
Chr20b
Chr20c
Chr22a
Chr22b
Els19
Esc32b
Kra30b
Lipa90b
Nug30
Sko56
Sko64
Sko72
Sko81
Sko90
Sko100a
Sko100b
Sko100c
Sko100d
Sko100e
Wil50


Memeticos Todos
Memeticos Subconjunto
Memeticos Mejores
Greedy
Tiempo
8
7
6
5
4
3
2
1
0
Chr20b
Chr20c
Chr22a
Chr22b
Els19
Esc32b
Kra30b
Lipa90b
Nug30
Sko56
Sko64
Sko72
Sko81
Sko90
Sko100a
Sko100b
Sko100c
Sko100d
Sko100e
Wil50

En media los memeticos-mejores tienen un mejor coste, sobretodo en instancias grandes.

En instancias pequeñas los que mejor funcionan son los meméticos-todos que hace un mayor número de busquedas locales al aplicarla sobre todos los cromosomas de la población. Es tambien por este motivo que tienen mayores tiempos (que se acentuan en instancias grandes) que los demás algoritmos.

Como es lógico en tiempo los mejores son los meméticos mejores, seguidos de los meméticos-subconjunto y por último meméticos-todos. Esto es debido al número de búsquedas locales que hacen.

También se puede observar (en los tres algoritmos) que cuanto mayores son las instancias, los costes estan mas cercanos al óptimo y los tiempos son mas altos.