Computacion evolutiva: Actividad 1

Jordi Colomer Matutano

December 14, 2012

1 Descripcion del problema

En esta actividad vamos a construir un algoritmo genético para resolver el problema del vendedor ambulante. El problema consiste en, dado un conjunto de ciudades y sus coordenadas, encontrar el camino más corto que pase por todas ellas. Para nuestros experimentos hemos utilizado un conjunto de datos disponible en internet en `http://www.tsp.gatech.edu/world/countries.html`. En concreto hemos utilizado los datos pertenecientes al Sahara Occidental. Éste está compuesto de 29 puntos o ciudades, y el óptimo global es conocido y es de distancia 27603. Tal y como se indica en la pagina web, el número óptimo de 27603 corresponde a la distancia mínima cuando ésta se calcula redondeando cada sector del camino al entero más cercano. Como nos conviene conocer el máximo global para la evaluación del programa, nosotros también haremos el cálculo de la distancia redondeando.

2 Implementación

Hemos programado un conjunto de clases en java que nos permiten implementar algoritmos genéticos de forma genérica y modular. En concreto tenemos una clase Main que es el punto de entrada del programa. Este lee un fichero de configuración donde está especificado el problema y la metodología para resolverlo. La clase Main és la encargada de instanciar todas las demás clases que componen los módulos del algoritmo genético. Por ultimo, se ejecutara el esquema básico de todos los algoritmos genéticos que hará las llamadas al resto de clases.

En nuestra implementación, hemos representado un camino para las n ciudades $C_0..C_n$, como una vector de enteros de dimensión n, donde cada elemento del vector indica una ciudad, y el orden del vector es el orden en que se visitan las ciudades,

conectando la primera y la ultima. Por ejemplo, el vector 2,4,1,3 representaría el camino que va de 2 a 4, de 4 a 1, de 1 a 3, y de 3 a 2.

Vamos a ver a continuación los distintos parámetros que se pueden definir en el fichero de configuración.

```
fitnessFunction=tsp
filename=wi29.tsp
```

Primero tenemos la especificación del problema, para ello solo nos es necesario especificar la clase que contiene la función fitness. En el caso de que esta corresponda a el problema del vendedor ambulante, se especificara otro parámetro con el nombre del fichero que contiene las ciudades y sus coordenadas. El fichero debe estar en el formato estandard tsp.

```
mu=40
lambda=20
maxGenerations=1000
```

La variable mu especifica el numero de individuos por generación. Lambda es el numero de nuevos individuos generados en cada generación y maxGenerations es el numero de generaciones.

```
parentSelectionMethod=fitnessProp|ranking|tournament|random
parentSelectionMethodTournamentk=5
```

Tenemos 4 posibles valores para el método de selección de padres, que corresponden a los 3 métodos estudiados (Selección proporcional al grado de adaptación, Selección por ordenación y selección por torneo) más uno inventado que simplemente hace selecciones aleatorias. Este ultimo método nos resultara útil para evaluar distintas configuraciones como veremos más adelante. En el caso de elegir el método tournament, debemos especificar un parámetro más para indicar la variable k, que es el numero de individuos que participaran el el torneo de selección.

```
survivorSelectionMethod=fitnessProp|tournament|ranking
                            |steadyStateWorst|steadyStateRandom
parentSelectionMethodTournamentk=5
```

Tenemos 5 métodos para seleccionar a los supervivientes, todos ellos estudiados en clase (Dos modelos estacionarios y los tres modelos inspirados en los utilizados en la

selección de padres). Hay un sexto método implícito en ellos, que es el Modelo Generacional. Este no es más que un caso concreto de el Modelo estacionario, para cuando los valores de lambda y mu coinciden. Por esta razón no lo programaremos implícitamente. De forma similar al caso anterior, también necesitamos un nuevo parámetro para indicar el valor de k en caso de seleccionar el método tournament.

```
recombinationMethod=MOX|PMX|CX|OX
```

Tenemos disponibles 4 métodos para hacer las recombinaciones. Todos ellos están diseñados para ser utilizados en individuos codificados mediante permutaciones. Los métodos son *Modified Order Crossover* (MOX), *Partially Mapped Crossover* (PMX), *Cycle Crossover* (CX) y *Order Crossover* (OX).

```
mutationMethod=swap|insert|shuffle|invert
mutationRate=.5
```

Por ultimo, hemos implementado 4 métodos de mutación, para individuos representados con permutaciones. Estas son la Mutación por intercambio (swap), Mutación por inserción (insert), Mutación por desordenación (shuffle) y Mutación por inversión (invert). Tenemos un parámetro más que nos especifica la probabilidad de mutación de cada nuevo individuo.

3 Evaluación

En este apartado vamos a evaluar distintas configuraciones del algoritmo. Para ello, ejecutaremos el programa un total de 100 veces por configuración. Cada ejecución devuelve el mejor individuo encontrado. Nos interesara saber cual es la media de todos los 100 mejores individuos encontrados en cada ejecución. A este valor lo llamaremos *Mean Best Fitness* o MBS.

Hemos definido el éxito de una ejecución dependiendo de si la distancia del mejor individuo encontrado es de menos del doble de la distancia de la solución optima. El *Sucess Rate* o Factor de Éxito sera el numero de ejecuciones con resultado éxito de las 100 lanzadas por prueba.

El *Average number of evaluations to a solution*, o numero medio de evaluaciones para una solución, es el numero de nuevos individuos creados en cada ejecución exitosa, antes de que esta pase a tener éxito, i.e. se encuentre el primer individuo con distancia menor al doble del optimo global.

Por último, mostraremos también una medida del número de veces de las 100 ejecuciones en las que se encuentra el número global optimo.

Table 1: Resultados de la ejecución del caso base con selecciones aleatorias

μ	λ	gen	parent	X	P_{mut}	mut	survivor	MBS	evals	success	opt
20	10	500	random	MOX	0.5	swap	random	74932.28	0	0	0

Table 2: Resultados de cada método de selección de padres

μ	λ	gen	parent	X	P_{mut}	mut	survivor	MBS	evals	success	opt
20	10	500	tournament	MOX	0.5	swap	random	36653.39	591.78	100	1
20	10	500	ranking	MOX	0.5	swap	random	62386.47	139.62	3	0
20	10	500	fitnessProp	MOX	0.5	swap	random	64338.01	46.21	1	0

3.1 Selección

Tenemos disponibles 4 métodos de selección de padres, y 6 métodos de selección de supervivientes. Dada la importancia de estas tareas (son las responsables de guiar todo el proceso de evolución) nos parece indicado que estos sean los primeros parámetros que probemos.

Empezamos eligiendo los métodos random para ambas selecciones. Los demás parámetros los hemos fijado con los valores que hemos creído más naturales, y no los cambiaremos en toda la sección. Los parámetros elegidos son MOX como método de recombinación, swap como método de mutación con probabilidad 0.5. La población contendrá 20 individuos, y en cada generación se generaran 10 nuevos individuos, en un total de 500 generaciones.

Esta primera prueba nos servirá como caso base, y la podemos utilizar como punto de referencia para el resto de ejecuciones. Los resultados los podemos ver en la tabla 1.

Como era de esperar, no ha habido ninguna ejecución con éxito. Hemos de tener en cuenta que dado que ambas selecciones son aleatorias, la función fitness no esta siento utilizada aún, y por tanto no hay nada que guíe a nuestro algoritmo hacia una buena solución.

A continuación probaremos todos los métodos disponibles para la selección de padres, dejando la selección de supervivientes a random. En la tabla 2 podemos ver los resultados.

Cabe destacar los excelentes resultados obtenidos con la selección de padres mediante torneo, que se encuentra de media solo a un 30% por encima de la solución optima. Incluso en una ejecución de las 100 probadas se encuentra la solución optima. Los dos otros métodos dan resultados parecidos, y no demasiado satisfactorios.

Ahora revertimos la selección de padres a random, y probamos cada uno de los

Table 3: Resultados de cada método de selección de supervivientes

μ	λ	gen	parent	X	P_{mut}	mut	survivor	MBS	evals	success	opt
20	10	500	random	MOX	0.5	swap	steadyStateWorst	37538.73	853.95	100	0
20	10	500	random	MOX	0.5	swap	tournament	37828.84	512.53	100	0
20	10	500	random	MOX	0.5	swap	ranking	42645.71	1452.26	99	0
20	10	500	random	MOX	0.5	swap	fitnessProp	58985.57	493.11	19	0
20	20	500	random	MOX	0.5	swap	generacional	72436.71	0	0	0

métodos disponibles para la selección de supervivientes. Los resultados los podemos ver en la tabla 3.

En este caso los dos mejores métodos, con resultados parecidos y también muy buenos son steadyStateWorst y tournament. El peor método es obviamente generacional, ya que este solo tiene sentido si tenemos un método de selección de padres que no sea random, en caso contrario nos encontramos de nuevo con que ningún método esta llamando a la función fitness, y por eso obtenemos un resultado parecido a la primera configuración con los dos métodos aleatorios.

Uno puede pensar que seguramente la mejor combinación podría ser utilizar el método tournament para la selección de padres y el steadyStateWorst para los supervivientes. La realidad es que la bondad de un método no es algo independiente, y es relativo a la selección del otro método. Es decir, sabemos que el método tournament en la selección de padres obtiene mejores resultados que los otros métodos si el método de selección de supervivientes es random, pero no podemos decir nada antemano si el método de selección de supervivientes es otro. Es por este motivo que para encontrar la mejor pareja nos vemos obligados a probar todas las 24 posibles combinaciones. En la tabla 6 listamos todas las ejecuciones. Como siempre las filas se encuentran ordenadas por el campo MBS.

La mejor combinación encontrada es tournament con generacional. Y como era de esperar, la peor es la prueba inicial con ambas selecciones random. La configuracion tournament y steadyStateWorst se encuentra en la quinta posición, que son los dos métodos óptimos cuando los buscamos de manera individual. Esto nos indica que encontrar los parámetros óptimos para un problema es una tarea compleja. Por simplicidad, en el resto del experimento solo ajustaremos los parámetros eligiendo de uno en uno.

El método tournament recibe un parámetro k, que en todos los experimentos anteriores hemos fijado a valor 5. En la tabla 5 mostramos diferentes ejecuciones para distintos valores de k. En concreto valores 3,5 y 7. En ningún caso mejoramos los resultados, por tanto dejaremos el valor inicial 5 en las sucesivas ejecuciones. De ahora en adelante

Table 4: Resultados de todas las posibles combinaciones de selecciones

μ	λ	gen	parent	X	P_{mut}	mut	survivor	MBS	evals	success	opt
20	20	500	tournament	MOX	0.5	swap	generacional	35083.91	572.64	100	0
20	10	500	ranking	MOX	0.5	swap	steadyStateWorst	36599.3	741.53	100	0
20	10	500	fitnessProp	MOX	0.5	swap	steadyStateWorst	36604.52	758.99	100	0
20	10	500	tournament	MOX	0.5	swap	random	36653.39	591.78	100	1
20	10	500	tournament	MOX	0.5	swap	steadyStateWorst	36937.48	475.21	100	0
20	10	500	ranking	MOX	0.5	swap	tournament	37156.73	503.22	100	0
20	10	500	random	MOX	0.5	swap	steadyStateWorst	37538.73	853.95	100	0
20	10	500	tournament	MOX	0.5	swap	tournament	37558.7	489.29	100	0
20	10	500	random	MOX	0.5	swap	tournament	37828.84	512.53	100	0
20	10	500	fitnessProp	MOX	0.5	swap	tournament	37829.16	513.5	100	0
20	10	500	tournament	MOX	0.5	swap	ranking	39038.39	745.93	100	0
20	10	500	ranking	MOX	0.5	swap	ranking	40272.93	1134.54	100	0
20	10	500	tournament	MOX	0.5	swap	fitnessProp	41135.08	1028.64	100	0
20	10	500	random	MOX	0.5	swap	ranking	42645.71	1452.26	99	0
20	10	500	fitnessProp	MOX	0.5	swap	ranking	42735.93	1560.21	99	0
20	10	500	ranking	MOX	0.5	swap	fitnessProp	53035.43	1803.3	65	0
20	10	500	fitnessProp	MOX	0.5	swap	fitnessProp	55921.92	1073.39	41	0
20	20	500	ranking	MOX	0.5	swap	generacional	58107.44	689.35	15	0
20	10	500	random	MOX	0.5	swap	fitnessProp	58985.57	493.11	19	0
20	20	500	fitnessProp	MOX	0.5	swap	generacional	60989.56	237.97	6	0
20	10	500	ranking	MOX	0.5	swap	random	62386.47	139.62	3	0
20	10	500	fitnessProp	MOX	0.5	swap	random	64338.01	46.21	1	0
20	20	500	random	MOX	0.5	swap	generacional	72436.71	0	0	0
20	10	500	random	MOX	0.5	swap	random	74932.28	0	0	0

Table 5: Diferentes valores de k para tournament (entre paréntesis)

μ	λ	gen	parent	X	P_{mut}	mut	survivor	MBS	evals	success	opt
20	10	500	tournament(3)	MOX	0.5	swap	steadyStateWorst	38196.24	873.6	100	0
20	10	500	tournament(5)	MOX	0.5	swap	steadyStateWorst	36937.48	475.21	100	0
20	10	500	tournament(7)	MOX	0.5	swap	steadyStateWorst	36787.55	469.29	100	0

Table 6: Ejemplos con elitismo

μ	λ	gen	parent	X	P_{mut}	mut	survivor	MBS (elite)	evals	success	opt	MBS (no elite)
20	10	500	random	MOX	0.5	swap	tournament	37124.36	598.37	100	0	37828.84
20	10	500	random	MOX	0.5	swap	fitnessProp	47899.72	2304.68	97	0	58985.57
20	10	500	random	MOX	0.5	swap	ranking	39787.72	1243.35	100	0	42645.71

dejaremos también fijados los dos métodos de selección.

Para terminar los métodos de selección, probaremos varias configuraciones de selección de supervivientes con elitismo. El concepto de elitismo solo tiene sentido para los métodos tournament, fitnessProp y ranking. En la tabla 6 mostramos los resultados para cada uno de los métodos, eligiendo random como selección de padres, y los comparamos con la versión sin elitismo. Para ello añadimos una columna con el MBS de la versión no elitista. Podemos ver fácilmente que la versión elitista es siempre mejor que la no elitista.

3.2 Recombinación

En esta sección probaremos los 4 distintos métodos que tenemos de recombinación. Para ello recuperaremos la mejor configuración encontrada hasta ahora, y le cambiaremos el método de recombinación hasta haberlos probado todos. En la tabla 7 veremos los resultados, donde se puede comprobar que el método OX obtiene mejores resultados que el método MOX utilizado hasta este momento. Los otros dos métodos PMX y CX dan resultados ligeramente peores. De ahora en adelante dejaremos el método de recombinación a OX.

Table 7: Métodos de recombinación

μ	λ	gen	parent	X	P_{mut}	mut	survivor	MBS	evals	success	opt
20	20	500	tournament	MOX	0.5	swap	generacional	35083.91	572.64	100	0
20	20	500	tournament	OX	0.5	swap	generacional	34048.6	499.8	100	0
20	20	500	tournament	PMX	0.5	swap	generacional	36693.62	558.46	100	0
20	20	500	tournament	CX	0.5	swap	generacional	36131.21	612.95	100	0

Table 8: Resultados de los experimentos seleccionados

μ	λ	gen	parent	X	P_{mut}	mut	survivor	MBS	evals	success	opt
20	20	500	tournament	OX	0.5	swap	generacional	34048.6	499.8	100	0
20	20	500	tournament	OX	0.5	insert	generacional	33153.13	463.66	100	2
20	20	500	tournament	OX	0.5	shuffle	generacional	39159.98	836.45	100	0
20	20	500	tournament	OX	0.5	invert	generacional	28405.46	369.88	100	4

Table 9: Resultados de los experimentos seleccionados

μ	λ	gen	parent	X	P_{mut}	mut	survivor	MBS	evals	success	opt
20	20	500	tournament	OX	0.3	invert	generacional	28606.11	384.9	100	7
20	20	500	tournament	OX	0.5	invert	generacional	28405.46	369.88	100	4
20	20	500	tournament	OX	0.7	invert	generacional	34544.43	408.91	100	0

3.3 Mutación

Hasta este momento hemos utilizado el método de mutación más sencillo, el método *swap*. En la tabla 8 comparamos este con los otros 3 métodos, utilizando la mejor configuración encontrada para el resto de parámetros.

Nos encontramos que de nuevo mejoramos los resultados cuando probamos con el método *insert*. De hecho esta configuración encuentra la solución optima 2 veces de 100 ejecuciones. Pero el método *invert* mejora los resultados más aún con un MBS muy cercano al óptimo, y encontrando este en 4 instancias de 100.

En la tabla 9 probamos diferentes probabilidades de mutación. En ningún caso obtenemos una mejora sobre la probabilidad escogida hasta el momento de 0.5.

3.4 Parámetros de población

Finalmente la tabla 10 muestra dos ejecuciones más, cambiando el tamaño de la población y el numero de generaciones. El incremento de población obtiene un MBS más bajo, y un numero de evaluaciones superior. Esto es una muestra de que el tamaño de la población es un parámetro importante, y que no es cierto que un valor mayor siempre produce mejores resultados. En cambio, el incremento de el numero de generaciones, si que siempre producirá mejores (o iguales) resultados, a cambio de un mayor tiempo de computación. En nuestro experimento la configuración con 1000 generaciones es la mejor configuración encontrada, con un MBS muy cercano al optimo, con el número medio de evaluaciones para obtener una solución exitosa más bajo obtenido y con un 7% de las ejecuciones devolviendo el resultado óptimo.

Table 10: Resultados de los experimentos seleccionados

μ	λ	gen	parent	X	P_{mut}	mut	survivor	MBS	evals	success	opt
20	20	500	tournament	OX	0.5	invert	generacional	28405.46	369.88	100	4
40	40	500	tournament	OX	0.5	invert	generacional	28410	564.33	100	7
20	20	1000	tournament	OX	0.5	invert	generacional	28267.36	369.88	100	7