TIA-Master:
Soluciones metaheurísticas para problemas complejos

Ramon Ruiz Dolz

Octubre 2018

Contents

1 Introducción

En esta practica se ha realizado la implementación de técnicas metaheurísticas para la resolución de problemas. Estas técnicas suelen aplicarse a problemas que no cuentan con un algoritmo o heurística específica que permita hallar una solución satisfactoria. Existen gran cantidad de técnicas metaheurísticas, sin embargo la mayor parte de estas siguen unos mismos patrones. Suelen estar inspirados en la naturaleza, basados en principios físicos o biológicos. Hacen uso de componentes estocásticos, haciendo uso de variables aleatorias. No hacen uso de la gradiente de la función objetivo y además, cuentan con parámetros que deben ser ajustados a mano en función del problema a resolver [2]. En este trabajo se ha realizado la implementación de un Algoritmo Genético y de Enfriamiento Simulado. A continuación se ha descrito el problema planteado en este trabajo.

1.1 Problema

El problema escogido para la aplicación de estas técnicas ha sido el problema del viajante de comercio o TSP (Traveling Salesman Problem [6]). Este problema plantea la situación en la que un comerciante desea visitar una serie de ciudades para vender sus productos. Únicamente desea visitar cada ciudad una vez y, claramente quiere que la ruta le sea lo menos costosa posible. El objetivo al resolver este problema consiste en hallar la ruta menos costosa para el comerciante, que pase una vez por cada ciudad y que termine en el mismo punto en el que empieza. Por lo tanto hablamos de un problema de optimización, concretamente de minimización. Este problema es de gran interés puesto que se encuentra dentro de los problemas NP-completos, lo que implica que hallar una solución eficiente para este problema implicaría encontrarla también para muchos otros problemas de esta categoría[10, 4].

Para el planteamiento de este problema existen gran cantidad de modificaciones como se expone en [1], sin embargo en este trabajo nos centraremos en la versión más simple de este, sin restricciones adicionales y con un único comerciante. En las siguientes secciones se han explicado los dos algoritmos implementados en este trabajo para abordar este problema.

2 Algoritmo Genético

En esta sección se ha definido la implementación del algoritmo genético realizada para la resolución del problema TSP. Un Algoritmo Genético [7] consiste en realizar una representación genética del problema a resolver. En primer lugar, las soluciones son representadas como cromosomas, hay que desarrollar un proceso de selección, otro de cruce, de mutación, etc. De forma genérica se suele inicializar una población de cromosomas (soluciones) sobre las cuales se aplica, en primer lugar un proceso de selección, normalmente en función del "valor" de esta. Para ello es importante definir la función *fitness* del problema. Mediante

este *fitness* ya se puede seleccionar la mejor, peor o soluciones intermedias. Después del proceso de selección, se aplica el cruce entre las distintas soluciones seleccionadas. Este cruce consiste en realizar combinaciones de distintas soluciones, las mas comunes pueden ser *n-point* o *uniform crossover*. Posterior al cruce se aplican mutaciones sobre la población. La mutación consiste en aportar aleatoriedad a la construcción de soluciones para evitar quedar estancados en un óptimo local. Finalmente se realiza la operación de reemplazo. Esta consiste en dar cabida a los nuevos cromosomas en la población activa y de retirar los cromosomas que ya no sean útiles. La función de reemplazo suele hacer uso del valor *fitness* de cada individuo para decidir quienes serán supervivientes y quienes no.

A continuación se ha explicado punto por punto cada uno de los elementos própios de un algoritmo genético aplicados al problema resuelto.

2.1 Población inicial

En primer lugar es importante, por una parte definir el cromosoma o la solución del problema. Para abordar el TSP existen multitud de posibilidades a la hora de definir los cromosomas. Existen propuestas [5] que plantean una lista de representación ordinal de las ciudades para evitar posteriores perdidas de factibilidad de la solución en la mutación. Sin embargo, en este trabajo se plantea el cromosoma como una lista literal de las ciudades en el orden a visitar. Cada cromosoma es, por lo tanto, una lista de longitud N, sea este el número de ciudades en cada instancia del problema.

La población inicial generada es una serie de cromosomas generados aleatóriamente en función de la talla del problema. En el caso de tener una N muy grande se trabajará con una población de tamaño más reducido para hacer computacionalmente asequible cada iteración. En cambio, en caso de N ser pequeña, se trabajará con una población considerablemente mayor con el objetivo de explorar un mayor número de soluciones por cada iteración.

2.2 Fitness

La función *fitness* se utilizará para definir la calidad de cada solución. En este trabajo se ha decidido hacer uso de la distancia euclídea entre dos ciudades. Para cada cromosoma, el cálculo de su valor de *fitness* consistirá en,

$$\textit{fitness(c)} = \sum_{n}^{N-1} \textit{distance}(n, n+1) \tag{1}$$

se defina la función *distance* como,

$$\textit{distance(x, y)} = \sqrt{\sum_{d=1}^{D} (x_d - y_d)^2} \tag{2}$$

sea X e Y las dos ciudades sobre las que se pretende medir la distancia y D el número de dimensiones de las coordenadas (típicamente 2).

Mediante esta función, por lo tanto, se puede obtener la calidad de una solución como el sumatorio de distancias euclídeas entre ciudades consecutivas en la lista de tamaño N. Dado que estamos ante un problema de minimización, será mejor solución la que menor valor de *fitness* tenga.

2.3 Selección

El principal objetivo de la función de selección consiste en determinar, de los miembros de la población actual, cuales van a ser elegidos para realizar el cruce entre ellos. Esta función toma como referencia el valor de *fitness* de cada cromosoma y, a partir de este elige a los candidatos.

En este trabajo se han implementado dos procesos de selección distintos, un proceso de selección elitista y un proceso de selección proporcional. El proceso de selección elitista únicamente tiene en cuenta los 10 mejores individuos de cada generación y sobre estos crea los nuevos individuos. En cambio, el proceso de selección proporcional tiene en cuenta el fitness propio del individuo y la media de fitness de la población, para asignar la probabilidad de ser seleccionado de un individuo. De esta forma, los individuos con mayor fitness tienen mayor probabilidad de ser escogidos y, por lo tanto de mejorar progresivamente la solución. Este método de selección puede tener un problema de convergencia prematura, sin embargo se tratará de paliar mediante la función de mutación.

2.4 Cruce

La función de cruce es la encargada de, a partir de los individuos seleccionados previamente, crear nuevos individuos mezclando sus datos. Mediante esta función se pretende obtener nuevos individuos resultado de la combinación de soluciones previas. Gracias a la función de cruce se puede conservar los tramos entre ciudades con valor *fitness* parcial bajo, y combinar estos tramos tratando de hallar la solución óptima. Como podemos observar en [8] existen gran cantidad de propuestas para esta función.

En este trabajo se hace uso de la técnica de cruce por n puntos con n = 1. Cada individuo es dividido en dos partes y se calcula el fitnes de cada una de ellas. Las partes con mejor fitness son combinadas en un individuo y las partes con peor fitness en otro individuo. En la función de cruce se puede dar que aparezcan ciudades repetidas. Para solventar este problema, únicamente hay que comprobar que no haya ninguna ciudad repetida. En caso de haberla, ésta se reemplaza por una de las ciudades que no se encuentran en la lista, repitiendo este proceso hasta tener una solución factible.

2.5 Mutación

En un algoritmo genético, la función de mutación tiene como objetivo evitar converjer en un mínimo local que no mejore. Esto es posible gracias al factor

aleatorio asociado a la mutación.

La función de mutación se ha implementado de manera que dado un cromosoma determinado, se realizarán $|N|/5$ permutaciones entre elementos aleatorios de la solución.

2.6 Reemplazo

A partir de, tanto la función de cruce como la de mutación, se han generado una serie de individuos nuevos. La función de reemplazo es la encargada de hacer espacio a estos nuevos individuos dentro de la población.

Existen múltiples propuestas de funciones de reemplazo. El reemplazo generacional, por ejemplo, únicamente permite a sus individuos permanecer una generación vivos. El remplazo por estado-estacionario, a diferencia del anterior, propone generar pocos hijos por generación y reemplazar ese número de individuos del subconjunto de peores individuos de la población. También existen propuestas catastrofistas, el juicio final por ejemplo consiste en salvar al mejor individuo y volver a generar aleatóriamente el resto de individuos.

En este trabajo se ha realizado un reemplazo por estado-estacionario. Se han realizado experimentos combinándolo con juicio final y sin combinar. De esta forma, se generan unos pocos nuevos individuos por generación y los peores son reemplazados. Además, en caso de no mejorar la última mejor solución encontrada, la población es destruida y generada aleatoriamente de nuevo. Se conserva únicamente la mejor solución y los nuevos individuos generados en esa iteración.

2.7 Resultado

El algoritmo se ha implementado respetando la naturaleza *any-time*. Puede ser detenido en cualquier momento y obtendremos la mejor solución hallada. Está programado para imprimir en pantalla la mejor solución encontrada hasta el momento cada vez que aparece una mejor.

3 Enfriamiento Simulado

El Enfriamiento Simulado [9] se basa en física estadística. Este se inspira en la técnica de enfriamiento utilizada en la metalurgia para obtener un estado sólido 'bien ordenado'. De esta forma, se eleva la temperatura del material y se trata de ir bajándola lentamente. En este caso, el material será la solución al problema y la temperatura se irá reduciendo iteración a iteración hasta hallar el máximo o mínimo global. A partir de una solución inicial generada aleatoriamente o heurísticamente, en cada iteración se escoge aleatoriamente una solución hijo a la solución actual (una variación de esta), y es aceptada en función de la temperatura. En el caso de que la solución mejore la anterior en un problema de minimización, la solución nueva es aceptada. En caso de que esto no sea así, la solución nueva se acepta con la probabilidad,

$$p(aceptar) = e^{\frac{f(new-sol)-f(sol)}{T}} \quad (3)$$

La temperatura va disminuyendo conforme avanza la búsqueda. Como podemos observar en la fórmula de arriba, al principio de la búsqueda existirá mayor probabilidad de aceptar soluciones que empeoren el valor de la solución actual. Para evitar devolver una solución que no sea la mejor hallada en el proceso de búsqueda, se suele almacenar la mejor solución encontrada hasta el momento.

En esta sección se ha definido la implementación del algoritmo de enfriamiento simulado realizado en este trabajo. Para ello en primer lugar hay que generar una solución inicial, posteriormente escoger una de sus soluciones vecinas, y finalmente decidir si reemplazar la solución actual o no. Todo ello mientras va disminuyendo la variable que mide la temperatura como se ha explicado anteriormente.

En este caso la solución inicial se ha generado de la misma forma que en el algoritmo genético. Se genera aleatoriamente una lista de talla N donde N es el número de ciudades existentes en la instancia del problema. Esta lista esta formada por el conjunto de números de [0..N-1] dónde cada número representa una ciudad y tiene una posición en el plano asociada.

A partir de esta solución, el algoritmo, en cada iteración genera un vecino aleatoriamente de una forma similar a la mutación del algoritmo genético. El vecino de la solución actual se genera intercambiando aleatoriamente dos ciudades de la lista. A continuación se comparan los valores de *fitness* de la solución actual con la nueva generada. En caso de que la nueva solución tenga un valor menor de *fitness* esta es aceptada inmediatamente (minimización). En caso de no mejorar el valor de *fitness* de la solución actual, la nueva solución es aceptada con la probabilidad definida anteriormente.

Al finalizar este proceso de selección, la temperatura T disminuye:

$$T = T/(i + k * T) \quad (4)$$

donde i es la iteración actual y k es una constante a la cual se le ha asignado el valor de 0.01.

Este algoritmo termina o bien al superar el número máximo de iteraciones o bien en caso de converger con la solución actual totalmente enfriada, T=0.

4 Casos de estudio

En este trabajo, trás realizar la implementación de ambos algoritmos, se han estimado 4 casos de estudio. Cada uno de ellos consta con una serie de características particulares como podemos observar a continuación. Las instancias del problema basadas en geografía real se han obtenido a partir de la web de la universidad de Waterloo[1]

[1] http://www.math.uwaterloo.ca/tsp/world/countries.html

4.1 Caso trivial

El primer caso de estudio consiste en un caso trivial. Este caso ha sido utilizado para comprobar que los algoritmos con capaces de hallar la solución óptima de forma correcta y que sus funciones están correctamente implementadas.

Para este caso de estudio se han utilizado 6 ciudades distintas ubicadas en los puntos (0,0), (1,0), (1,1), (1,2), (0,2), (0,1) dentro de un espacio de representación bidimensional. La solución óptima a este problema es de 6.

4.2 WSahara

Una vez comprobado que los algoritmos eran capaces de encontrar la solución óptima a problemas triviales se ha considerado el uso de ejemplos 'reales'. El primero de estos es WSahara. Este dataset esta formado por 29 ciudades distintas inspiradas en la geografía del Sahara. La solución óptima a esta instancia es de 27603.

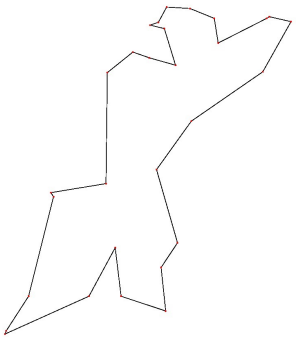

Figure 1: Ruta óptima al problema de WSahara

En la Figura 1 podemos observar gráficamente esta solución óptima.

4.3 Djibouti

En segundo lugar se ha escogido Djibouti, un problema compuesto por 38 ciudades inspiradas en la geografía de este país. La solución óptima a esta instancia del problema es 6656.

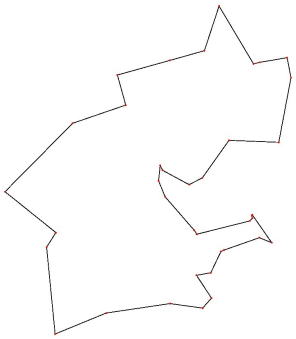

Figure 2: Ruta óptima al problema de Djibouti

En la Figura 2 podemos observar gráficamente esta solución óptima.

4.4 Qatar

Finalmente se ha utilizado el dataset de Qatar. Este set de datos esta formado por 194 ciudades ubicadas en localizaciones própias de Qatar. La solución óptima a esta instancia es de 9352.

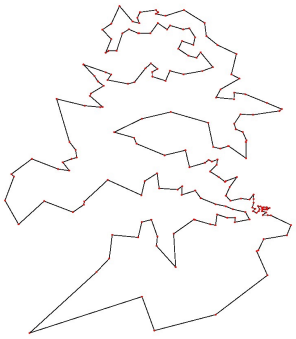

Figure 3: Ruta óptima al problema de Qatar

En la Figura 3 podemos observar gráficamente esta solución óptima.

5 Resultados obtenidos

Sobre las instancias del problema descritas en la sección anterior se han realizado dos experimentos. En primer lugar una comparativa entre las distintas implementaciones del algoritmo genético y en segundo lugar una comparativa, entre la mejor versión del algoritmo genético y el enfriamiento simulado. Se han lanzado 5 ejecuciones por cada dataset y algoritmo modificando la *seed* para que

se generasen números aleatorios diferentes. Para cada experimento hay que remarcar que se ha utilizado la misma semilla, siendo esta únicamente modificada en las ejecuciones. En las siguientes tablas se pueden observar las diferencias de resultados entre las distintas implementaciones del algoritmo genético. En primer lugar se comparan los resultados sobre el caso de estudio trivial:

Dataset	**Juicio Final**	**Selección**	**Iteraciones**	**Fitness**
Trivial	No	Elitista	87	24.45109
Trivial	No	Proporcional	620	24.45109
Trivial	Sí	Proporcional	620	24.45109

Table 1: Resultados en Trivial

Como podemos observar, en este caso de estudio no existe ninguna diferencia. Esto es debido a la simpleza de esta instancia del problema. En segundo lugar podemos observar las diferencias encontradas en el dataset WSahara:

Dataset	**Juicio Final**	**Selección**	**Iteraciones**	**Fitness**
WSahara	No	Elitista	90316	50341.15091
WSahara	No	Proporcional	43108	53561.12475
WSahara	Sí	Proporcional	43099	53561.12475

Table 2: Resultados en WSahara

En este caso ha sido la versión elitista del algoritmo la capaz de hallar la mejor solución a la instancia del problema. A continuación tenemos las diferencias encontradas en el dataset Djibouti:

Dataset	**Juicio Final**	**Selección**	**Iteraciones**	**Fitness**
Djibouti	No	Elitista	74871	15083.43912
Djibouti	No	Proporcional	88187	17256.59834
Djibouti	Sí	Proporcional	82815	16764.77590

Table 3: Resultados en Djibouti

En esta tercera parte del primer experimento, ya se pueden apreciar más diferencias. En este caso se ha obtenido un mejor resultado para la versión elitista, un segundo puesto para la proporcional con juicio final y el pero resultado ha sido para la versión proporcional sin juicio final. Esto puede deberse a la convergencia prematura del algoritmo. Mediante el juicio final es posible evitar o huir de esa convergencia. Finalmente, la última parte de este experimento consiste en lanzar las ejecuciones sobre el dataset Qatar. A continuación se pueden observar los datos obtenidos:

Dataset	Juicio Final	Selección	Iteraciones	Fitness
Qatar	No	Elitista	82308	69245.68206
Qatar	No	Proporcional	6356	73837.93677
Qatar	Sí	Proporcional	70235	69966.84656

Table 4: Resultados en Qatar

Al finalizar esta ejecución podemos observar de nuevo las mismas características y particularidades que en la anterior. El mejor resultado se consigue con el elitista, el segundo mejor con el proporcional con juicio final y el peor con la versión sin juicio final.

A pesar de estos resultados, para realizar el segundo experimento se ha considerado el uso del algoritmo genético con selección proporcional y juicio final. Aunque los resultados hayan sido ligeramente mejores por parte de la versión elitista del algoritmo, hay un gran factor de aleatoriedad en este tipo de algoritmos. Como se puede observar en [3] dónde se ha realizado un análisis más exhaustivo al respecto, el método elitista no termina de ser fiable, es por esto que se ha deicidido trabajar con la selección proporcional con juicio final.

En la siguiente tabla se puede observar la media de iteraciones y de valor de *fitness* obtenido al finalizar las 40 ejecuciones finales en las cuales se compara el rendimiento del algoritmo genético y el enfriamiento simulado:

Dataset	Algoritmo	Iteraciones	Fitness
Trivial	Algoritmo Genético	96	24.45109
Trivial	Enfriamiento Simulado	76	24.45109
WSahara	Algoritmo Genético	43099	53561.12475
WSahara	Enfriamiento Simulado	4565	40259.53458
Djibouti	Algoritmo Genético	82815	16764.77590
Djibouti	Enfriamiento Simulado	3934	9694.29668
Qatar	Algoritmo Genético	70235	69966.84656
Qatar	Enfriamiento Simulado	297718	18215.1966

Table 5: Resultados obtenidos tras la experimentación

Se puede observar como el rendimiento del enfriamiento simulado para los experimentos realizados en este trabajo ha sido contundentemente superior. En las siguientes subsecciones se han analizado los resultados obtenidos en mayor detalle para cada *dataset* y se han comparado los datos obtenidos.

5.1 Caso trivial

El experimento con el dataset trivial tiene un coste computacional muy bajo. La media de tiempo de ejecución para ambos algoritmos no ha alcanzado los 5 segundos para ninguno de ambos casos. Usando este dataset podemos observar que el número de iteraciones medias empleadas por cada algoritmo es muy

próximo. Además se puede observar también que en todos los experimentos se ha alcanzado la solución óptima con ambos algoritmos.

5.2 WSahara

El experimento con el *dataset* WSahara ya ha permitido observar las primeras diferencias entre algoritmos. Dado el tamaño de este dataset, tampoco ha implicado un coste computacional muy elevado. Se puede observar que existe una diferencia de 38534 entre ambos algoritmos. En este experimento el promedio de iteraciones empleadas para encontrar la mejor solución antes de cumplir cualquiera de las condiciones de parada le da la victoria al enfriamiento simulado puesto que ha realizado un menor número de iteraciones. Por otra parte, analizando el valor *fitness* medio conseguido, se puede observar que mediante el enfriamiento simulado también se alcanza una solución de mayor calidad antes de detener el algoritmo.

5.3 Djibouti

El experimento con el *dataset* Djibouti presenta algunas diferencias respecto al anterior. En este caso, el dataset es ligeramente mayor en tamaño respecto al anterior, sin embargo sigue siendo de talla tratable. El promedio de iteraciones requeridas para hallar la mejor solución antes de detenerse dista en 78881 iteraciones. En este caso el algoritmo genético ha tardado un mayor número de iteraciones en encontrar su mejor solución. Esto se traduce en que, mediante Enfriamiento Simulado se ha conseguido una media de *fitness* muchísimo más próxima a la solución óptima que el algoritmo genético en un menor tiempo de cómputo. De hecho la media de las soluciones del algoritmo genético son de una calidad bastante moderada.

5.4 Qatar

Finalmente, el experimento realizado con el *dataset* Qatar. Este dataset esta compuesto de 194 ciudades, esta talla del problema ya lo complica notablemente. El promedio de iteraciones requerido para hallar la mejor solución antes de detenerse en este caso se dispara. La diferencia entre el número de iteraciones es de 227483, siendo el genético otra vez el que más iteraciones ha requerido para alcanzar su mejor solución antes de detenerse. Sin embargo en este experimento concretamente, el algoritmo genético ha funcionado particularmente regular. El algoritmo de enfriamiento ha alcanzado soluciones próximas a la óptima, sin embargo el genético ha triplicado el valor *fitness* del ES. Esto en un problema de minimización implica que esa solución es notablemente peor. Además, cabe destacar que el algoritmo genético en este experimento también ha tardado un mayor tiempo en terminar debido a su mayor complejidad algorítmica y a la talla de esta instancia del problema.

6 Conclusiones

Para cerrar esta memória me gustaría enfatizar en las siguientes conclusiones. El hecho de haber implementado los algoritmos desde cero ha implicado una serie de búsquedas y lecturas que han permitido ampliar mi entendimiento acerca de este tipo de algoritmos. Por otra parte, se ha podido comprobar como existe facilidad a la hora de programar los algoritmos, pero indeterminación a la hora de escoger los parámetros. Encontrar los mejores parámetros para cada problema puede consistir en una tarea realmente compleja. Por lo tanto, tomando los experimentos realizados en este trabajo, los resultados obtenidos con el algoritmo de Enfriamiento Simulado son notablemente mejores. Sin embargo, no se puede afirmar rotundamente que este algoritmo sea el mejor para este tipo de problema en todos los casos, únicamente en los casos experimentados en este trabajo.

References

[1] Tolga Bektas. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3):209–219, 2006.

[2] Ilhem BoussaïD, Julien Lepagnot, and Patrick Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82–117, 2013.

[3] Chetan Chudasama, SM Shah, and Mahesh Panchal. Comparison of parents selection methods of genetic algorithm for tsp.

[4] Michael R Garey, David S. Johnson, and Larry Stockmeyer. Some simplified np-complete graph problems. *Theoretical computer science*, 1(3):237–267, 1976.

[5] Rajeev Gopal, Brian Rosmaita, and Dirk Van Gucht. Genetic algorithms for the traveling salesman problem. In *Proceedings of the first International Conference on Genetic Algorithms and their Applications*, pages 160–168, 2017.

[6] Karla L Hoffman, Manfred Padberg, and Giovanni Rinaldi. Traveling salesman problem. In *Encyclopedia of operations research and management science*, pages 1573–1578. Springer, 2013.

[7] John H Holland. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence, 1975.

[8] Abid Hussain, Yousaf Shad Muhammad, M Nauman Sajid, Ijaz Hussain, Alaa Mohamd Shoukry, and Showkat Gani. Genetic algorithm for traveling salesman problem with modified cycle crossover operator. *Computational intelligence and neuroscience*, 2017, 2017.

[9] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[10] Christos H Papadimitriou. The euclidean travelling salesman problem is np-complete. *Theoretical computer science*, 4(3):237–244, 1977.