Máster Universitario en Investigación en Inteligencia Artificial

Resolución de problemas con metaheurísticos.

Comparación de un algorítmo poblacional y uno de trayectoria (problema de la mochila).

Autor
Omar Suárez López

28 de febrero de 2018

1. Introducción.

En el presente trabajo se van a presentar los resultados obtenidos de la comparativa realizada entre dos algorítmos metaheurísticos, uno de trayectoria (*Simulated Annealing*) y otro poblacional (*Genetic Algorithm*). El primer metaheurístico ha sido programado en Python por mí, mientras que el *GA* es una implementación en Java obtenida de la web de la *UMA* a la que ha sido necesario implementarle el código para ejecutar el problema de la mochila.

1.1. Problema de la mochila (*knapsack problem*).

El problema de la mochila o *knapsack problem* es muy conocido en el campo de la optimización y ha sido utilizado en decenas de papers para comparativas con diferentes tipos de algorítmos. Se trata de un problema donde se tiene una serie de objetos o productos los cuáles tienen un cierto valor y un peso. En su versión más simple se pretende introducir en una mochila aquellos objetos que maximicen el valor total de lo que se guarde dentro pero con la limitación de un peso máximo. La formulación matemática es la siguiente,

$$\begin{aligned} &\text{maximize} \sum_{i=1}^{n} v_i x_i \\ &\text{subject to} \sum_{i=1}^{n} w_i x_i \leq W \text{ and } x_i \in \{0,1\} \end{aligned} \tag{1}$$

Siendo v_i el valor de cada objeto, x_i la selección o no del objeto i (valor 0 o 1), w_i el peso del objeto i y W el peso máximo que la mochila es capaz de admitir (restricción).

1.2. Instancia del problema.

Para la comparativa de los algorítmos *GA* y *SA* se va a utilizar una instancia de tamaño medio obtenida de la *OR library*. En el cuadro 1 se muestran los parmetros del problema seleccionado.

Parámetro	Cantidad
Objetos	50
Mochilas (restricciones)	5

Cuadro 1: Tamaño de la instancia del problema de la mochila.

Como la cantidad de objetos es de 50 la solución viene representada por un vector de 1s y 0s de 50 posiciones y a su vez las 5 mochilas nos impondrán las restricciones de peso que la solución no podrá sobrepasar.

2. Métodos de evaluación para algorítmos metaheurísticos.

Para evaluar y comparar ambas metaheurísticas se ha tomado la decisión de utilizar la calidad de la solución obtenida en la ejecución del algorítmo siendo el valor a comparar finalmente calculado mediante la fórmula $|f(s) - f(s^*)|/f(s^*)$, en la cuál s es la solución obtenida y s^* es el óptimo global conocido de la instancia del problema. Los pasos planteados para llevar a cabo la comparativa se resumen en los siguientes,

1. *Tuning* de los parámetros necesarios para cada algorítmo (30 ejecuciones con cada combinación de parámetros estableciendo valores pequeño-mediano-grande).
2. Selección de los mejores parámetros en función de la media y desviación estándar en cada combinación.
3. Realización de otras 30 ejecuciones con los parámetros óptimos.
4. Comprobación de la normalidad de los resultados (test de *Kolmogorov-Smirnov*)
5. Selección del test de hipótesis adecuado a la normalidad o no de los resultados:
   a) Normalidad: *Student's t-test*
   b) No normalidad: test de *Wilcoxon*

3. Manejo de las restricciones.

En los problemas de optimización donde es necesario manejar restricciones (como es el caso del problema de la mochila) es necesario definir la manera en que estas van a ser gestionadas por el algorítmo. Hay varias aproximaciones para su manejo (estrategias de eliminación, estrategias de penalización, estrategias de reparación, etc) y su elección dependerá de diferentes factores (tipos de soluciones obtenidas, tipo de problema abordado, etc). En este caso se ha optado, por simplicidad, por presentar una **estrategia de penalización** del siguiente tipo,

$$f'(x) = \begin{cases} f(x), & \text{si } x \text{ factible} \\ 0, & \text{si } x \text{ no factible} \end{cases}$$

En el caso entonces de que la solución obtenida durante el proceso de búsqueda sea no factible (no supera alguna de las restricciones impuestas) y siendo este un problema de maximización, se asigna un valor de 0 a la función de coste de dicha solución (en caso de un problema de minimización se haría al contrario, se asignaría un valor muy elevado a la función de coste). Para realizar una comparativa justa este criterio se utiliza en ambas metaheurísticas.

4. Afinación de parámetros.

4.1. Afinación de parámetros del *Simulated Annealing*.

En el caso del *Simulated Annealing* se necesita establecer una estrategia de enfriamiento lo más óptima posible para el funcionamiento del algorítmo. Los parámetros que se necesita definir son:

- Temperatura inicial
- Estado de equilibrio
- Función de enfriamiento
- Criterio de parada

4.1.1. Temperatura inicial.

Hemos seleccionado una estrategia *Accept all* en este caso, estableciendo una temperatura inicial muy alta de manera que se acepten la mayoría de vecinos en las primera iteraciones.

4.1.2. Estado de equilibro.

Para tratar de lograr un estado de equilibrio durante el paso por cada una de las temperaturas se opta por establecer una estrategia *estática* en la cuál determinamos las transiciones antes del comienzo de la búsqueda. El número de transiciones se debe determinar en base al tamaño del vecindario. Como esto requiere un estudio previo hemos seleccionado, para cada ejecución, un valor

adecuado al criterio de parada (número de evaluaciones de la función de coste) de forma que en cada prueba se consiga que el algorítmo lo cumpla.

4.1.3. Función de enfriamiento.

En esta implementación del *Simulated Annealing* se ha optado por utilizar una función de enfriamiento geométrica dada por la fórmula $T = \alpha T$.

4.1.4. Criterio de parada.

Para el criterio de parada se opta por el número de evaluaciones de la función de coste, de manera que se pueda establecer una comparativa justa entre ambos algorítmos (el número de evaluaciones en ambos es igual). Otro de los criterios más utilizados en el *Simulated Annealing* es establecer una temperatura mínima, de tal forma que se pare al llegar a ella. Aquí hemos establecido el valor de una temperatura mínima pero no se utiliza realmente como criterio de parada.

El cuadro 2 resume los valores de los parámetros para cada una de las pruebas del afinado del *SA*.

Parámetro	Ejecución 1	Ejecución 2	Ejecución 3
Temperatura inicial	5000	8000	8000
Iteraciones para cada temperatura	1000	2000	3000
Temperatura final	0.01	0.01	0.01
Parámetro α	0.9	0.8	0.7
Total de iteraciones	100000	100000	100000

Cuadro 2: Parámetros para pruebas de afinado del *SA*

4.2. Afinación de parámetros del *Genetic Algorithm.*

Los parámetros que vamos a necesitar afinar para el algorítmo *GA* van a ser,

- **Probabilidad de mutación**: las mutaciones representan cambios pequeños en individuos seleccionados entre toda la población. La probabilidad de mutación (p_m) nos indica la probabilidad de mutación de cada elemento (*gene*) de la solución. Generalmente se recomienda el uso de valores pequeños ([0.001, 0.01]).
- **Probabilidad de cruce**: el uso del operador *crossover* es que la nueva solución herede algunas características de sus padres para generar la nueva prole.

Estos dos parámetros serán los que intentemos afinar en las primeras pruebas antes de realizar la comparativa mediante los test de hipótesis. El cuadro 3 resume los valores de los parámetros para su afinado,

Parámetro	Ej. 1	Ej. 2	Ej. 3	Ej. 4	Ej. 5	Ej. 6	Ej. 7	Ej. 8	Ej. 9
Probabilidad de mutación	0.3	0.3	0.3	0.6	0.6	0.6	0.9	0.9	0.9
Probabilidad de cruce	0.3	0.6	0.9	0.3	0.6	0.9	0.3	0.6	0.9

Cuadro 3: Parámetros durante cada ejecución para pruebas de afinado del *GA*.

5. Resultados de la afinación.

Para la obtención de los parámetros óptimos mediante el afinado se han realizado 30 ejecuciones para cada configuración presentada en las tablas del apartado anterior. Los resultados finales basados en la media y la desviación estándar de las soluciones obtenidas en las 30 ejecuciones nos dán como parámetros óptimos para el *SA*,

- Temperatura inicial = 8000
- Iteraciones para cada temperatura = 2000
- Temperatura final = 0.01
- $\alpha = 0.8$
- Total de iteraciones = 100000

En el caso del *GA* los parámetros que mejor resultados han dado son,

- Probabilidad de cruce = 0.3
- Probabilidad de mutación = 0.3

Todos las pruebas realizadas y los datos se encuentran disponibles en el archivo *tabla_datos_pruebas.xlsx* para su consulta.

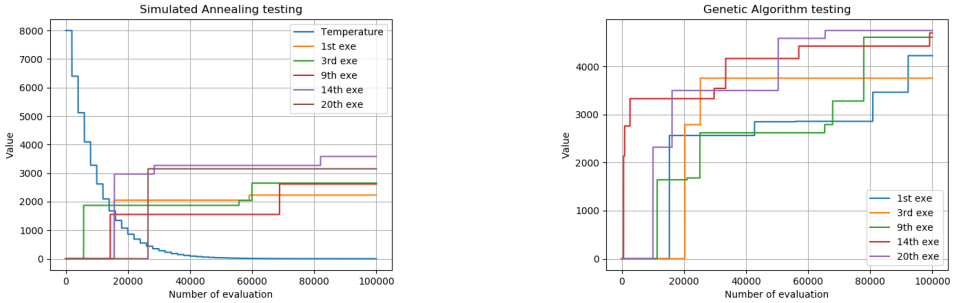
6. Comparativa.

Una vez seleccionados los parámetros más prometedores para la instancia hacemos un mínimo de 30 ejecuciones de nuevo de cada algorítmo obteniendo los valores presentados en el cuadro 4.

Ejecución	Simulated Annealing	Genetic Algorithm
1	0.6376	0.3138
2	0.4015	0.3182
3	0.5695	0.3898
4	0.4214	0.3140
5	0.5086	0.2175
6	0.4620	0.3166
7	0.5314	0.5012
8	0.5900	0.3378
9	0.5746	0.2516
10	0.4010	0.3386
11	0.5767	0.4070
12	0.4680	0.3464
13	0.5111	0.2568
14	0.4179	0.2368
15	0.4192	0.2662
16	0.4781	0.2506
17	0.3528	0.2708
18	0.5349	0.2735
19	0.3081	0.3219
20	0.4882	0.2289
21	0.4424	0.2091
22	0.4596	0.2693
23	0.4900	0.3383
24	0.2443	0.2685
25	0.4146	0.3404
26	0.1946	0.2992
27	0.4367	0.3188
28	0.3141	0.2247
29	0.3300	0.2906
30	0.4979	0.1522
media	0.4492	0.2956
σ	0.1029	0.0675

Cuadro 4: Resultados de las 30 ejecuciones con los parámetros óptimos.

Las imágenes de la figura 1 nos muestran la evolución de la solución obtenida en función de las evaluaciones de la función de coste para las ejecuciones realizadas con los parámetros óptimos. Se han seleccionado diferentes ejecuciones para su representación en las gráficas de manera que podamos tener una visión aproximada de cómo se desarrolla la busqueda del óptimo dentro de cada algorítmo. En la figura 1a podemos ver además la evolución de la temperatura con $\alpha = 0{,}8$ para el SA.



(a) Simulated Annealing (b) Genetic Algorithm

Figura 1: Evolución de las soluciones en función del número de evaluaciones de la función de coste.

6.1. Normalidad.

Para una posterior selección del test de hipótesis comprobamos si los datos obtenidos cumplen normalidad mediante el test de *Kolmogorov-Smirnov*. Este test está disponible en varias bibliotecas de R. Los resultados nos dan un *p-value* muy inferior a 0.05 tanto en la distribución del *SA* como en el *GA* lo que indica la **no normalidad** de los mismos. El test a aplicar para la comparativa en caso de no normalidad es el test de **Wilcoxon**, un test no paramétrico muy utilizado en varios papers de comparativas de metaheurísicos.

6.2. Resultados finales.

Como se comentaba en el apartado anterior el test de hipótesis que se utiliza para la comparativa es el *test de Wilcoxon*. La **hipótesis nula** en este caso es que ambas metaheurísticas tiene el mismo comportamiento. En caso de que el *p-value* sea menor que el umbral de significancia establecido (0.05) se descartará la hipótesis nula en pro de la hipótesis alternativa (los resultados son significativamente diferentes). Para el computo del test hemos utilizado la biblioteca de R y la función *wilcox.test* con la opción *paired=TRUE*. El resultado del *p-value* es $p - value = 1{,}63E - 07$ (muy por debajo del umbral 0.05) lo que nos hace rechazar la hipótesis nula y aceptar que los **resultados obtenidos entre ambas metaheurísticas son significativamente diferentes**. Volviendo a los resultados aportados en el cuadro 4 claramente el algorítmo *GA* supera con mucho el rendimiento del *SA* presentado.

Referencias

[1] Talbi, El-Ghazali. Metaheuristics, from design to implementation.

[2] Jason Brownlee. Clever Algorithms.

[3] Luke, Sean. Essentials of Metaheuristics: A set of undergraduate lecture notes.

[4] Metaheuristics for bussines Analytics, Chapter 2, General concepts in metaheuristic search.

[5] Gendreau, Michel, Potvin, Jean-Yves. Handbook of Metaheuristics.

[6] ssGA: Steady State GA
`http://neo.lcc.uma.es/software/ssga/index.php`