Estudio práctico de un algoritmo genético básico sobre el problema de la mochila

Javier Martínez Álvarez

Master Universitario en Investigación en Inteligencia Artificial
Universidad Internacional Menendez Pelayo
100006269@alumnos.uimp.es
https://github.com/jaluma/RPCM-KnapsackGA

Abstract. La importancia de los operadores de mutación y cruce en un algoritmo evolutivo (EA) es muy crucial para la búsqueda de la solución óptima. En este estudio se ha evaluado la importancia de los operadores de mutación y cruce en un algoritmo. Partiendo de una implementación de un algoritmo genético de estado estacionario ya desarrollado [1], se ha implementado la función *fitness* correspondiente al problema de la Mochila 0-1. Usando las instancias mknap1, se han hecho diferentes pruebas para determinar la eficiencia de los operadores de mutación y cruce. A su vez, se ha buscando obtener un estudio más en profundidad sobre la busqueda del óptimo y la detención de la busqueda al superar el número máximo de iteraciones.

Keywords: algoritmos · metahurísticos · genetico · mochila · ssga

1 Introdución

1.1 Problema de la mochila

El problema de la mochila es un problema de optimización combinatoria, en el que se desea encontrar una solución óptima para una mochila de capacidad que contiene un conjunto de objetos, cada uno de los cuales tiene un peso.
Inicialmente, la mochila se encuentra vacía, y se desea maximizar el beneficio obtenido en la selección de los objetos. La solución al problema será la suma de los beneficios de los objetos que se incluyan en la mochila. Por tanto, podríamos definir matemáticamente el problema de la mochila como:

$$\max_{x \in 0,1} \sum_{j=1}^{n} p_j \cdot x_j \tag{1}$$

$$\ni x \in 0,1 \sum_{j=1}^{n} \sum_{i=1}^{m} r(i,j) \cdot x_j <= b(i) \tag{2}$$

Donde n es es el número de objetos que se incluyen en la mochila, m es el número de mochilas, x_j es el valor del gen, p_j es el beneficio que aporta el objeto, $r(i,j)$ es el peso del objeto y b_i es el peso máximo de la mochila.

1.2 Algoritmos genéticos

El problema de la mochila se puede resolver mediante un algoritmo genético. Este algoritmo consiste en una población de individuos, cada uno de los cuales es una solución parcial de la mochila. Se evalúa la aptitud de cada individuo mediante la función de fitness.

1.3 Operadores de cruce

Los operadores de cruce consisten en unir dos individuos, mediante el cruce, para formar un nuevo individuo. En nuestro estudio, se hará uso del operador de recombinación *Single Point Crossover* [2] (SPX). Este consiste en copiar una serie de genes de un individuo padre en un individuo hijo, y otra serie de genes de otro individuo padre en otro individuo hijo.
La partición se determina generando un número aleario y determinando si es inferior a la probabilidad de cruce. En caso afirmativo, realizará el cruce y generará el nuevo individuo. En caso contrario, se devolverá uno de los individuos padres.

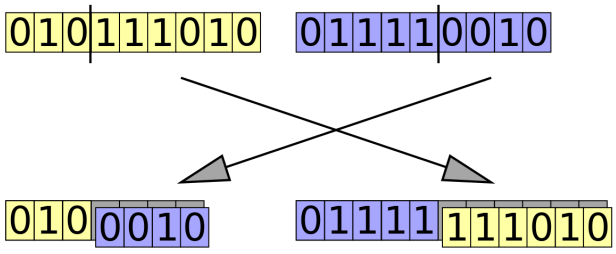

Fig. 1. Operador de cruce SPX

1.4 Operadores de mutación

Los operadores de mutación consisten en modificar un individuo modificando uno de sus genes heredados de alguno de sus padres. A partir de una probabilidad de mutación dada por el usuario, se determina si se realizará la mutación y se intercambiará el valor de uno de los genes por el contrario (torneo binario).
Considerando la probabilidad de mutación del problema, haremos uso de la siguiente operación:

$$p_m = \frac{1}{g_n \cdot g_l} \tag{3}$$

1.5 Algoritmo genético *Steady State* (ssGA)

ssGA es una implementación de un algoritmo genético de estado estacionario. Con los siguientes pasos de forma iterativa:

1. Se genera una población inicial de individuos.
2. Se evalúa el *fitness* de cada individuo de la población.
3. Se seleccionan dos individuos haciendo uso de un **torneo binario** ([3]).
4. Usando el operador de cruce **SPX** ([2]), se genera un nuevo individuo.
5. Al nuevo individuo, se le aplica el operador de mutación binaria.
6. Evaluar el *fitness* del nuevo individuo.
7. Se reemplaza el nuevo individuo en la población por el individuo que obtuvo el menor valor de *fitness*.

2 Estudio experimental

Para realizar el sigueinte estudio experimental, se ha usado las instancias para el problema de la mochila que se pueden encontrar en la siguiente [5]. Se ha definido un programa que nos permite leer la instancia y ejecutar el algoritmo genético para todas los problemas de la instancia. Estos problemas fueron definidos en el siguiente [4] *paper*.
Para comenzar, se ha decidido ejecutar cien iteraciones a cada uno de los siete problemas descritos en la entrada del problema, obteniendo un total de setecientas iteraciones. Para ello se ha buscando un equilibrio del tamaño de la población y el número de generaciones máximas. Por tanto, se ha configurado el algoritmo con los siguientes parámetros de ejecución:

- **Tamaño de la población**: 512
- **Número máximo de generaciones**: 50000
- **Número de repeticiones** 30
- **Porcentaje de cruce**: rango de cuatro números comprendido en $0.5--0.9$.
- **Modificador del porcentaje de mutación**: producto cuyo valor está en el rango de cuatro números comprendidod en $0.8--1.0$.

Una vez ejecutado el programa sobre los siete problemas, se ha obtenido una tabla con los resultados obtenidos (Tabla 1).

Se pueden obtener las siguientes conclusiones:

1. Los problemas más pequeños son resueltos perfectamente por el algoritmo con el 100% eficacia.
2. Comenzamos a obtener una eficacia inferior al 100% en el problema nº 4 pero muy cercana a la solución óptima (*fitness* de 6060 versus 6120).
3. La desviación típica de los resultados es cero en problemas pequeños haciendose mayor a medida que vamos aumentando el tamaño de la población en cada iteración.

Teniendo en cuanta los anteriores resultados, se ha decidido profundizar más realizando un estudio en la busqueda del óptimo y un estudio de la detención de la convergencia.

Resultados	Min ft	Media ft	Max ft	D. Típica
Problema 1	3800.00	3800.00	3800.00	0.0000
Problema 2	8706.10	8706.10	8706.10	0.0000
Problema 3	4015.00	4015.00	4015.00	0.0000
Problema 4	6060.00	6118.60	6120.00	4.9978
Problema 5	12300.00	12391.04	12400.00	14.8815
Problema 6	10375.00	10564.04	10618.00	41.0814
Problema 7	16311.00	16455,09	16537.00	50.1717

Table 1. Resultados de los problemas.

2.1 Estudio en la busqueda del óptimo

En dicha subsección, se va a hacer un analisis de los resultados obtenidos en busca de la solución óptima. Para ello, se va a reducir el analisis a los siguientes parámetros:

- Se realizará sobre los problemas 4 y 5.
- Porcentaje de cruce: rango de dos elementos comprendido en $0.5 - -0.6$.
- Modificador del porcentaje de mutación: rango de dos elementos comprendidos en $0.8 - -1.0$.

A partir de estos datos, se ha analizado como afecta los porcentajes de cruce y mutación en la eficacia del algoritmo. Inicialmente se estudia el primero. A razón de las 30 muestras y de las 480 combinaciones que se han realizado, una resultados más que aceptables son:

- **Porcentaje de cruce**: 0.77
- **Modificador de la mutación**: 0.3 ó 0.4
- **Eficacia**: 0.7
- **Iteraciones medias**: 20263

Haciendo uso de ambos modificadores, hemos obtenido una eficacia en la obtención de la solución óptima del 70%. El algoritmo tiene un buen rendimiento con este problema, dando soluciones óptimas y rápidas en pocas iteraciones ($\tilde{7}518$) pero viendose lucrado por la eficacia.
Una posible alternativa, con mucho mayor eficacia, sería tomar una combinación de parámetros que nos ha arrojado un 100% de eficacia en el problema 4.

- **Porcentaje de cruce**: 0.63
- **Porcentaje de mutación**: 0.03
- **Eficacia**: 1.0
- **Iteraciones médias**: 7563

Tal como se puede observar, los resultados son perfectos, encontrando siempre la solución óptima pero tendiendo a necesitar alguna iteración más para alcanzarla, con respecto a la anterior solución. Si analizamos de forma gráfica el resultado obtenido(Figura 2), se puede observar que el algoritmo ha convergido

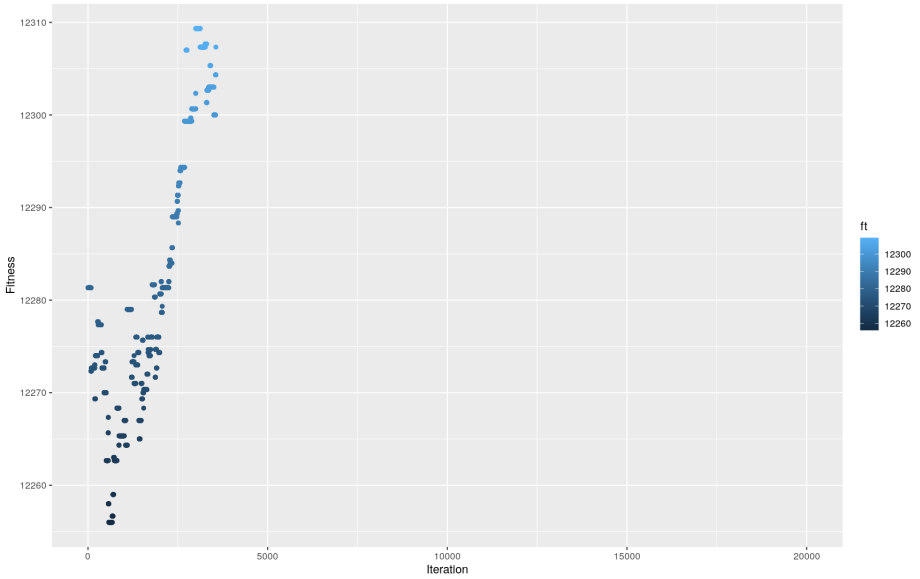

Fig. 2. Progreso con un porcentaje de cruce 0.63 y un porcentaje de mutación de 0.03.

a la solución óptima en la iteración nº 7563 como máximo.
Inicialmente, se obtiene un resultado relativamente bueno, las próximas iteraciones se ve inmerso en una busqueda local hasta que en la iteración $\tilde{5}00$ empieza a mejorar de forma notable.
En resumen, la primera solución nos aporta la ventaja de que el algoritmo tiene un buen rendimiento manteniendo un buen rango de eficacia a costa de necesitar menor iteraciones para alcanzar un solución cercana a la óptima. Por otro lado, la alternativa que se ha tomado es la que nos ha arrojado una eficacia más alta, con un número de iteraciones más alto de media, pero una menor dispersión en el número de iteraciones necesarias para alcanzar la solución óptima.

2.2 Estudio en la detención de la convergencia

En este apartado se va a tratar de analizar el comportamiento del algoritmo en el caso de que el problema sea un problema lo suficientemente grande que dificulte la convergencia en la solución óptima. Para ello, se va a reducir el analisis a los siguientes parámetros:

- Se trabajará sobre el problema más complejo, el nº 6.
- Porcentaje de cruce: rango de dos elementos comprendido en $0.5 - -0.6$.
- Modificador del porcentaje de mutación: rango de dos elementos comprendidos en $0.8 - -1.0$.

Partiendo de los datos obtenidos en la ejecución máxima descrita con anteriorad en 2.1, se ha observado que el algoritmo tiende a converger si alguno

de los parámetros de cruce y mutación busca una solución local que coincide con la óptima. Visualizamos este comportamiento en la Figura 3. En el gráfico observamos la media de la *fitness* en las diferentes iteraciones. En las primeras iteraciones, hay un par de pruebas que se aproximan muy rápido a la solución óptima. A medida de que alcanzan la solución, vamos comprobando que la dispersión se mantiene estable en varios rangos, por ejemplo, [15500, 16000]. Como se puede comprobar, el algoritmo intenta mejorar en cada iteración realizando saltos escalonados en el valor final de la *fitness*. A medida que las iteraciones van avanzando, la dispersión aumenta en las zonas de soluciones locales, y se ve reducido en las zonas de soluciones óptimas y peores soluciones.

De tal analisis podemos deducir que un algoritmo evolutivo tiende a converger en una solución óptima o muy cercana a ella.

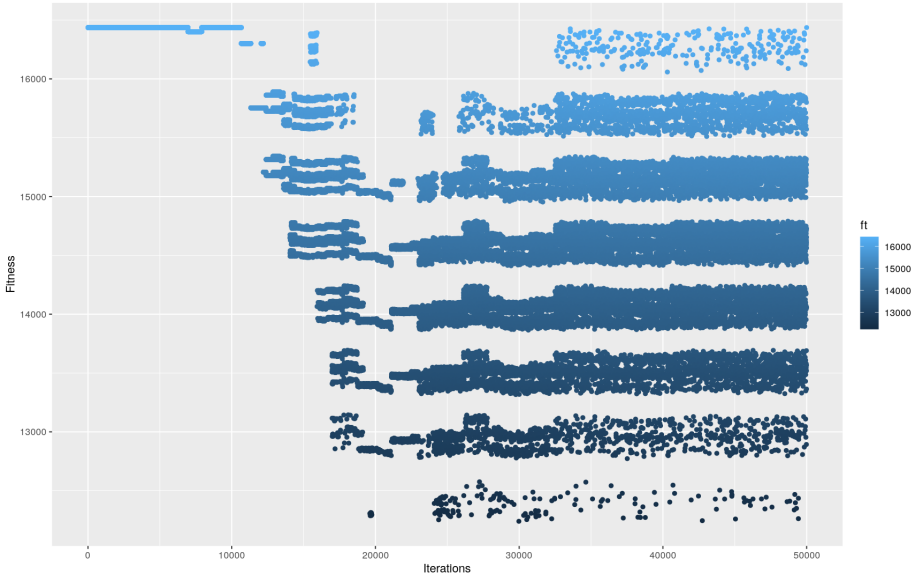

Fig. 3. Progreso con un porcentaje de cruce 0.5 y un porcentaje de mutación de 0.016.

3 Conclusiones

En conclusión, haciendo balance de los datos y pruebas realizadas en los apartados anteriores, se puede decir que, los algoritmos genéticos son una herramienta muy útil para la resolución de problemas, como el problema de mochila 0-1. Después de analizar y poder comparar los resultados de multiples pruebas, vemos unos resultados realmente buenos. Todas las soluciones obtenidas son óptimas o muy cercanas a la óptima.

Evidentemente, se podría intentar mejor y/o acercarse más a la solución óptima cuando no hemos obtenido los mejores resultados, pero hay que encontrar un punto equilibrio entre rendimiento y eficacia. dado que con problemas grandes, el tiempo de computo se dispara para prácticamente no identificar soluciones mejores a las encontradas. Por otro lado, sería importante recalcar la importancia que tiene los parámetros que nos permiten configurar el algoritmo. Una mala selección de ellos, nos puede llevar a una solución que no sea óptima. Dado el analisis experimental hecho, podemos determinar que porcentaje de cruce y de mutación bajos nos han arrojado mejores resultados, siendo en muchos casos la solución óptima.
Para terminar, se podría decir que los algoritmos genéticos nos permiten resolver problemas complejos de forma muy eficiente, de forma aproximada, teniendo un mucho mejor desempeño que un algoritmo exacto.

References

1. Author, E., Author: Steady State Genetic Algorithm (ssGA), (1999). https://neo.lcc.uma.es/software/ssga/index.php
2. Author, T., Author, S., Author, M.: Theoretical Analysis of Simplex Crossover for Real-Coded Genetic Algorithms, France (2000). http://www.springer.com/engineering/computational-systems-control/theoretical-analysis-of-simplex-crossover-for-real-coded-genetic-algorithms
3. Tournament selection, (2021). https://en.wikipedia.org/wiki/Tournament_selection
4. Author, C. C. :Computational experience with variants of the Balas algorithm applied to the selection of R&D projects, (1967). https://en.wikipedia.org/wiki/Tournament_selection
5. Author, J., Author: OR Library, http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html