# Knapsack Problem solucionado con múltiple enfoques

Jesús Marcano
Pedro Samuel Fagundez
Universidad Simón Bolívar

## Resumen

Esta publicación tiene como objetivo analizar el rendimiento de múltiples implementaciones de algoritmos para solucionar el 0/1 Knapsack Problem (el problema de la mochila donde un objeto puede o no estar, y nunca está múltiples veces). Estos algoritmos se sometieron a un dataset de pruebas común múltiples veces, lo cual permitió dejar en evidencia clara el rendimiento de dichos algoritmos para específicamente solucionar el problema.

Alguno de los algoritmos implementados fueron: Algoritmos Genéticos, Colonia De Hormigas, Local Search, entre otros.

**Keywords:** GA, Algoritmos Genéticos, Local Search, Knapsack Problem, KP, Problema de la Mochila, Algoritmo Memético, Colonia de Hormigas.

## Introducción

Los problemas de optimización en los que las variables de decisión son enteras, es decir, donde el espacio de soluciones está formado por ordenaciones o subconjuntos de números naturales, reciben el nombre de problemas de optimización combinatoria.

El problema de la mochila, comúnmente abreviado por **KP** (del inglés *Knapsack Problem*) es un problema de optimización combinatoria, es decir, que busca la mejor solución entre un conjunto finito de posibles soluciones a un problema. Modela una situación análoga al llenar una mochila, incapaz de soportar más de un peso determinado, con todo o parte de un conjunto de objetos, cada uno con un peso y valor específicos. Los objetos colocados en la mochila deben maximizar el valor total sin exceder el peso máximo.

Para esta publicación, se trato el problema 0/1 en el que un objeto está o no está, y no puede estar múltiples veces.

Las técnicas utilizadas fueron heurísticas y metaheurísticas, métodos diseñados para resolver un problema cuando se ignora si la solución es correcta, pero que usualmente produce un resultado suficientemente bueno o resuelve un problema similar pero más sencillo o que se interseca con la solución de problemas más complejos.

Así mismo, todos los algoritmos fueron implementados por nosotros, por lo tanto dichas implementaciones también se encuentran abiertas a mejoras.

## 1. Algoritmos empleados

Para resolver el KP se utilizaron los siguientes métodos/algoritmos:

- Programación dinámica
- Local Search
- Tabú Search
- Algoritmos Genéticos
- Algoritmos Meméticos
- Colonia de Hormigas
- Metaheurística “el que tenga miedo a morir, que no nazca”.

Algunos de estos algoritmos emplean soluciones similares con intentos de mejoras de algoritmos semejantes, en algún caso lo que se desea es observar si los intentos de mejora empeoran o realmente mejoran el rendimiento del algoritmo.

Con relación a la búsqueda local se implementaron tanto la versión original como la iterativa, para este problema, obviamos la implementación iterativa, ya que es solo una variación de la búsqueda local original y su rendimiento fue mucho peor.

## 2. Detalle de las implementaciones

### 2.1 Local Search

Iniciamos el algoritmo escogiendo 1 solución inicial completamente aleatoria.

El espacio combinatorio E utilizado para nuestra búsqueda local son las distintas combinaciones de selección de objetos dentro de la mochila que respetan que la suma de sus pesos no puede superar el peso maximo.

La funcion de evaluacion toma una de las selecciones (una mochila) y retorna el valor total.

La vecindad es 1-intercambio, tomar uno de los elementos seleccionados y cambiarlo, 1 a la vez, para generar una nueva mochila.

### 2.2 Algoritmos Genéticos

La implementación de este algoritmo se basa en representar una mochila como un cromosoma, cada cromosoma (que es muy similar a un estado), posee un conjunto de genes que representan cuales objetos de las lista de objetos han sido colocados dentro de la mochila (1’s y 0’s). Estos cromosomas necesitan ser cruzados para comenzar a producir las soluciones. Dicho cruce ocurre de forma aleatoria seleccionando 2 cromosomas y cruzando sus genes, la forma de hacerlo es combinando la mitad de sus genes (lado izquierdo de uno combinado con el lado derecho del otro), generando 1 nuevo cromosoma “hijo”. Cómo último recurso, algunos hijos tienen permitida la mutación, donde con una cierta probabilidad es capaz de mutar algunos de sus genes.

Como previamente se mencionó de forma ligera, hacemos uso de un conjunto de parámetros ajustables para el algoritmo donde nos permite alterar la probabilidad de mutación y de cruce, para este caso se utilizó 10% de mutación y 90% de cruce.

El algoritmo cuenta con un límite de iteraciones que en este caso fue 10000.

**2.3 Algoritmo Memético**

Esta implementación es muy similar a la de Algoritmos Genéticos, con un par de ligeras diferencias que trata de mejorar el cruce entre cromosomas y la generación de nuevos hijos.

Esta nueva creación de hijos optimizada consiste en seleccionar 3 cromosomas aleatorios al momento del cruce en vez de 2, y se crea un hijo de la permutación de los genes de los 3, luego, como un proceso de optimización final para robustecer la solución, se ejecuta una búsqueda local sobre los genes del nuevo hijo generado.

Para aclarar, los parámetros de mutación y cruce se mantienen.

**2.4 Colonia de Hormigas**

Esta metaheurística fue implementada usando como representación un arreglo de feromonas (valores entre 0 y 1) y un arreglo de objetos (1's y 0's), el primer arreglo es la probabilidad que tiene cada hormiga por preferir tomar un objeto y el segundo los objetos tomados.

Las feromonas se inician con un muy bajo porcentaje a todas por igual, para lograr un mayor impacto de preferencia cuando un camino parezca una solución, ya que las feromonas son actualizadas en base a si una hormiga determina que tomar ese objeto en particular es parte de su solución encontrada.

Los objetos se toman por una combinación de probabilidades entre las feromonas y la relación peso valor de cada objeto.

El objetivo es que las hormigas descarten objetos que están lejos de ser solución y solo usen los que parecen solución, aunque por probabilidades siempre van a ser capaces de tomar un objeto que no es parte de los que se han vuelto los "preferidos". A su vez, a medida que un objeto se repite como solución para una hormiga, las otras hormigas aprenden de ello y también prefieren ese objeto.

Es importante mencionar también, que para evitar que soluciones viejas que fueron olvidades sigan teniendo una probabilidad elevada se implementa una optimización de "evaporación" que permite que todas las feromonas vayan decrementando su valor si no son utilizadas.

En este algoritmo existen parámetros ajustables de igual manera, que son el número de hormigas y el índice de evaporación de las feromonas, que en este caso utilizamos 100 y 10%. El algoritmo cuenta con un límite de iteraciones que en este caso fue 4000.

**2.5 Tabú Search**

La implementación de este algoritmo es exacta al de búsqueda local, la principal modificación es que algunas mochilas son almacenadas en una lista tabú o "prohibidos" para evitar ciclar sobre las mismas soluciones y explorar más soluciones diferentes.

Al comienzo del algoritmo encontramos una solución inicial, dicha solución se agrega a la lista tabú y a partir de ahí vamos generando soluciones a partir de esa solución, cada una de las soluciones nuevas es agregada a la lista tabú. La lista tiene un límite de tamaño la mitad de los objetos disponibles en el universo, y a partir de ese límite se remueve la primera solución de la cola para ir agregando las nuevas soluciones a la lista tabú. El algoritmo cuenta con un límite de iteraciones que en este caso fue 10000.

**2.6 Programación Dinámica**

En este caso utilizamos la implementación óptima de Geeks for Geeks.

**2.7 El que tenga miedo a morir, que no nazca**

Esta metaheurística fue creada por los autores de esta publicación, utiliza la frase "el que tenga miedo a morir que no nazca" como inspiración para dividir los elementos en dos grupos según su relación valor/peso y un umbral.

El umbral es un valor que sirve de criterio para medir la calidad de un elemento en el problema de la mochila.

Los elementos "fearless" tienen una alta relación valor/peso y se priorizan en la selección, mientras que los elementos "cautious" se consideran solo si queda espacio disponible en la mochila después de seleccionar los elementos "fearless".

La implementación consiste en encontrar un conjunto de elementos que maximice la ganancia total de la mochila sin exceder su capacidad a través de dos arreglos que son críticos para la búsqueda, un arreglo de elementos “fearless” (sin miedo) y otro de elementos “cautious” (temerosos).

Los elementos fearless son los elementos que superan el umbral y son considerados las mejores opciones para ser seleccionadas.

Los elementos fearless son los elementos que se encuentran por debajo del umbral y son considerados las opciones complementarias.

Con la implementación actual no se garantiza que el algoritmo evite converger tempranamente en soluciones subóptimas. La heurística actual es simple y está basada en la relación valor/peso de los elementos, lo que puede conducir a soluciones subóptimas en algunos casos. La ventaja de la metaheurística actual es su simplicidad y rapidez en la búsqueda de soluciones aproximadas.

Es importante tener en cuenta que la implementación de estas técnicas puede aumentar la complejidad del algoritmo y, por lo tanto, afectar su rendimiento en términos de tiempo de ejecución. En general, existe un equilibrio entre la calidad de las soluciones obtenidas y el tiempo de ejecución del algoritmo, y se debe encontrar el balance adecuado según las necesidades y restricciones del problema específico.

## Resultados

**Tabla 1: 250 Objetos disponibles en el universo.**

| 250 | | | | | | | | | | | Promedio | Maximo | Mediana | Desviacion Estandar |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| t DP | 0,002 | 0,003 | 0,006 | 0,003 | 0,001 | 0,001 | 0,001 | 0,001 | 0,001 | 0,001 | 0,002 | 0,006 | **0,001** | 0,0016 |
| v DP | 845 | 829 | 912 | 805 | **982** | 805 | 758 | 840 | 882 | 851 | 850,9 | 982 | 842,5 | 62,6852 |
| t LS | 0,002 | 0,002 | 0,001 | 0,001 | 0,001 | 0,001 | 0,001 | 0,001 | 0,001 | 0,001 | 0,0012 | 0,002 | **0,001** | 0,0004 |
| v LS | 601 | 790 | 819 | 658 | 799 | 677 | 665 | 613 | 809 | **895** | 732,6 | 895 | 733,5 | 101,2304 |
| t TS | 1,814 | 2,042 | 2,046 | 1,944 | 1,835 | 1,741 | 1,833 | 1,794 | 2,05 | 2,262 | 1,9361 | 2,262 | **1,8895** | 0,1621 |
| v TS | 292 | 294 | 409 | 288 | **466** | 350 | 228 | 236 | 292 | 294 | 314,9 | 466 | 293 | 74,0427 |
| t GA | 5,966 | 1,813 | 1,659 | 2,755 | 6,129 | 2,921 | 1,811 | 1,289 | 1,987 | 3,333 | 2,9663 | 6,129 | **2,371** | 1,7425 |
| v GA | 781 | 716 | 771 | 793 | 730 | 636 | 723 | 799 | 803 | **856** | 760,8 | 856 | 776 | 61,2931 |
| t MEME | 0,445 | 0,984 | 0,556 | 0,505 | 0,511 | 0,239 | 0,318 | 0,741 | 0,724 | 0,544 | 0,5567 | 0,984 | **0,5275** | 0,2160 |
| v MEME | 409 | 427 | **523** | 375 | 416 | 466 | 395 | 426 | 453 | 408 | 429,8 | 523 | 421 | 41,9915 |
| t HORMIGA | 0,782 | 0,843 | 0,756 | 0,752 | 0,905 | 0,731 | 0,922 | 0,908 | 1,002 | 0,855 | 0,8456 | 1,002 | **0,849** | 0,0893 |
| v HORMIGA | 760 | **850** | 677 | 742 | 803 | 755 | 623 | 682 | 835 | 701 | 742,8 | 850 | 748,5 | 73,1722 |
| t MH | 0,0001 | 0,0001 | 0,001 | 0,001 | 0,0001 | 0,001 | 0,0001 | 0,0001 | 0,0001 | 0,0001 | 0,00037 | 0,001 | **0,0001** | 0,0004 |
| v MH | 832 | 829 | 909 | 796 | 975 | 800 | 752 | 834 | 879 | 840 | 844,6 | 975 | 833 | 63,0665 |

**Tabla 2: 500 Objetos disponibles en el universo.**

| 500 | | | | | | | | | | | Promedio | Maximo | Mediana | Desviacion Estandar |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| t DP | 0,002 | 0,002 | 0,006 | 0,003 | 0,001 | 0,001 | 0,001 | 0,001 | 0,001 | 0,001 | 0,0019 | 0,006 | **0,001** | 0,0016 |
| v DP | 1047 | 1033 | 966 | 988 | 999 | **1087** | 870 | 1081 | 1046 | 1084 | 1020,1 | 1087 | 1039,5 | 67,2978 |
| t LS | 0,005 | 0,003 | 0,001 | 0,001 | 0,001 | 0,001 | 0,001 | 0,001 | 0,001 | 0,001 | 0,0016 | 0,005 | **0,001** | 0,0013 |
| v LS | 835 | 817 | **1009** | 752 | 891 | 565 | 680 | 829 | 840 | 779 | 799,7 | 1009 | 823 | 119,3902 |
| t TS | 8,186 | 8,707 | 8,623 | 8,675 | 9,452 | 9,331 | 8,558 | 9,435 | 8,855 | 9,093 | 8,8915 | 9,452 | **8,781** | 0,4225 |
| v TS | 294 | 295 | **412** | 236 | 295 | 354 | 295 | 410 | 411 | 295 | 329,7 | 412 | 295 | 62,6206 |
| t GA | 10,553 | 9,531 | 4,196 | 5,427 | 8,55 | 27,987 | 7,229 | 9,001 | 15,223 | 8,301 | 10,5998 | 27,987 | **8,7755** | 6,7987 |
| v GA | 754 | 810 | 766 | 849 | 747 | 857 | 721 | 819 | **917** | 830 | 807 | 917 | 814,5 | 60,1221 |
| t MEME | 1,034 | 1,633 | 1,043 | 1,241 | 1,707 | 0,611 | 1,294 | 1,039 | 1,091 | 1,636 | 1,2329 | 1,707 | **1,166** | 0,3450 |
| v MEME | 449 | 460 | 406 | 404 | 407 | 544 | 549 | 458 | **587** | 451 | 471,5 | 587 | 454,5 | 65,7018 |
| t HORMIGA | 3,424 | 1,497 | 1,233 | 1,305 | 1,841 | 3,102 | 1,126 | 1,203 | 1,283 | 2,035 | 1,805 | 3,424 | **1,401** | 0,8250 |
| v HORMIGA | 840 | **901** | 732 | 793 | 857 | 742 | 698 | 711 | 877 | 813 | 796,4 | 901 | 803 | 72,5568 |
| t MH | 0,0001 | 0,001 | 0,0001 | 0,0001 | 0,0001 | 0,0001 | 0,0001 | 0,0001 | 0,001 | 0,0001 | 0,00028 | 0,001 | **0,0001** | 0,0004 |
| v MH | 1032 | 1033 | 948 | 988 | 998 | 1073 | 870 | 1069 | 1046 | 1084 | 1014,1 | 1084 | 1032,5 | 65,9452 |

t: Representa una medida de tiempo en segundos.

v: Representa la mejor solución encontrada para esa corrida.

---

## Discusión de los resultados

Los resultados se encuentran dentro de lo esperado, DP es el método más rápido y que atina a la solución óptima. Luego, el resto suele tener un comportamiento recurrente con respecto a sus resultados.

En el caso de los algoritmos genéticos y colonia de hormigas se aproximan bastante a ser igual de buenos pero con tiempos demasiado elevados en comparación al DP.

La mediana de los tiempos nos demuestra que GA suele ser el más tardío por lo que colonia de hormigas es mejor que GA.

Tabú Search y el algoritmo memético son los más deficiente para resolver KP, las mejoras de la lista tabú y la generación de hijos a través de permutaciones son demasiado costosas y agregan un nivel de profundidad que no termina siendo una optimización adecuada para el problema.

Nuestra metaheurística presenta un comportamiento bastante interesante, aunque no alcanza en el 100% de los casos el mejor valor, si lo encuentra en muchas de las oportunidades, y en todos los escenarios se aproxima a un nivel bastante cercano, y lo más interesante es que lo hace más rápido que el DP.

Ambas tablas (250 y 500 objetos) se comportan de forma uniforme con respecto a los datos estadísticos, así que podemos decir

que no difiere el rendimiento con respecto a una diferencia importante de elementos.

---

## Conclusiones

La programación dinámica es el método óptimo para solucionar el 0/1 Knapsack Problem, como se pudo observar tanto en la tabla 1 como la 2, posee el valor máximo encontrado en las múltiples corridas.

La segunda opción, es nuestra metaheurística, que aunque no encuentra en el 100% de los casos la solución óptima, si lo hace en un mejor tiempo.

Tanto el algoritmo genético como el colonia de hormigas les siguen muy de cerca puesto que en algunos casos encontraron una solución muy cercana al DP, el único problema es que el costo de tiempo suele ser muchas veces más alto en comparación al DP.

Probablemente el mayor fallo de la tabú search es que al explorar constantemente nuevos caminos y la generación de local search ser aleatoria esto hace que explore demasiados caminos y requiera más iteraciones para lograr un resultado eficiente.

Desde nuestra perspectiva, creemos que a través de un modelo de aprendizaje y una estrategía de adaptación podríamos encontrar parámetros para ejecutar los algoritmos tales que podamos aprovechar al máximo su potencial, pero esto no pudo ser explorado por nuestra parte.

Finalmente, si se requiere conocer con exactitud la solución del KP es necesario utilizar DP, en caso de que solo sea necesario un aproximado, es mejor utilizar nuestra metaheurística original.

---

## Recomendaciones

Para mejorar las implementaciones se mencionan algunas mejoras que se pueden realizar:

1. En la inicialización de la solución inicial de local search se debe utilizar un greedy algorithm para aumentar el rendimiento del algoritmo en su búsqueda de la mejor solución.

2. En el algoritmo de las hormigas, notamos que una mejora interesante sería dinámicamente ordenar los objetos en base a su porcentaje de feromonas para que sean considerados de primero los más importantes, pero el ordenamiento debe ser ejecutado en un momento preciso (o cada n iteraciones) para evitar ciclar en las mismas soluciones.

3. Para nuestra metaheurística original “El que tenga miedo a morir, que no nazca” se pueden agregar mejoras para evitar una convergencia temprana. Algunas de las mejoras serían técnicas para aumentar la diversidad en la búsqueda de soluciones tales como: Introducir aleatoriedad en la selección de elementos, Reinicialización periódica,

Búsqueda Local y ajuste adaptativo del umbral.

---

## Referencias


1. Michel Gendreau, Jean-Yves Potvin, Handbook of Metaheuristics, 2010.
2. https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/
3. Láminas del curso CI-5652 del profesor Ricardo Monascal.