Problema de la mochila

Ma. Alicia López Ll.[1], Maira Paola Pereira[2], Claudia Valenzuela[3],
Juan Cáceres Flor[4]
Universidad Nacional de Asunción, Facultad Politécnica
Ingeniería Informática
{[1]malolla.lopez, [2] pitupereira, [3]claudiavalenzuela2, [4]carloscaceres.jf}@gmail.com

Resumen. El problema de la mochila puede ser resuelto a través de implementaciones de algoritmos tales como Voraz, las Vegas y Backtracking, cuyos resultados reflejan que el algoritmo voraz resulta el más óptimo debido que su heurística permite la obtención de buenos resultados en un tiempo despreciable, aunque dicho resultado no siempre es el óptimo, se aproxima bastante a él.

Palabras Clave: Mochila, Algoritmo, Voraz, Vegas, Backtracking.

1 Introducción

El problema de la mochila es uno de los 21 problemas NP-completos de Richard Karp, establecidos por el informático teórico en un famoso artículo de 1972.1 Ha sido intensamente estudiado desde mediados del siglo XX y se hace referencia a él en el año 1897, en un artículo de George Mathews Ballard.

Si bien la formulación del problema es sencilla, su resolución es más compleja. Algunos algoritmos existentes pueden resolverlo en la práctica para casos de un gran tamaño [2].

2 Descripción del Problema

El problema de la mochila, comúnmente abreviado por KP (del inglés Knapsack problem) es un problema de optimización combinatoria. Modela una situación análoga al llenar una mochila, incapaz de soportar más de un peso W, con todo o parte de un conjunto N de objetos *j*, cada uno con un peso w_j y valor v_j específicos [1], [2].

El problema de la mochila es definido formalmente como [1]:

$$max \sum v_j$$

$$\text{tal que } \sum w_j \leq W$$

La formulación más común del problema es la del llamado *problema de la mochila 0-1*. El problema es llamado 0-1 porque cada objeto puede ser enteramente aceptado o rechazado. En *Fractional Knapsack Problem*, puede tomarse una fracción del objeto de manera a completar el peso de la mochila [1].

3 Algoritmos Implementados

En el presente trabajo se analizarán soluciones para el *problema de la mochila 0-1*, para lo cual se han utilizado los algoritmos: 1) Voraz, 2) Las Vegas y 3) Bactracking, implementados en lenguaje Java, dichos algoritmos se detallan en las siguientes Subsecciones.

3.1 Algoritmo de las Vegas

Consiste en seleccionar un elemento como "semilla" a partir del cual se crea una solución posible de manera aleatoria considerando el peso. Dicha solución es almacenada en caso de ser mejor a la solución anterior, considerando su valor, pues esto se repite con todos los elementos disponibles.
Este algoritmo retorna la mejor solución encontrada aunque no asegura que sea la mejor.

3.2 Algoritmo Voraz

La utilización de un algoritmo voraz consiste en introducir en la mochila según orden decreciente de utilidad (beneficio) los diversos objetos. Se adicionarán unidades enteras hasta que, por motivo de capacidad, no sea posible seguir introduciendo elementos.

Concepto de solución óptima: si se ordenan los objetos de forma de decreciente en cuanto a su relación (valor/peso = v_j/w_j) y se introducen en la mochila en este orden mientras quepan, de esta manera, un elemento muy caro no se agrega si pesa demasiado en comparación con otro de menor valor, pero de mucho menor peso.

El nombre voraz proviene de que, en cada paso, el algoritmo escoge el mejor "pedazo" que es capaz de "comer" sin preocuparse del futuro. Nunca deshace una decisión ya tomada: una vez incorporado un candidato a la solución permanece ahí hasta el final; y cada vez que un candidato es rechazado, lo es para siempre.

En cuanto al desempeño de este método, su complejidad es la equivalente al proceso de ordenación de la lista de elementos por su valor por unidad de peso, que es O(N log N), ya que el proceso de llenado de la mochila es lineal.

3.3 Backtracking

Para obtener la solución al problema de la Mochila utilizando Backtracking, se utiliza un método recursivo encargado de revisar todas las alternativas, en donde se utiliza una mochila que irá almacenando los elementos que satisfagan las restricciones además de una mochila temporal que almacena la mejor solución encontrada hasta el momento. La implementación se inicia en el primer elemento donde se evalúa la posibilidad de agregar o no el elemento a la mochila y ejecutando recursivamente al método hasta alcanzar el peso máximo. Una vez que la mochila se llena, ésta, se compara con la mejor solución encontrada hasta ese momento.

Este método devuelve la solución óptima, puesto que evalúa todas las posibilidades, sin embargo, cuando el número de elementos disponibles es relativamente grande, la cantidad de operaciones a realizar hace prácticamente imposible utilizar el algoritmo, ya que su complejidad es del orden de O(N!), donde N es la cantidad de elementos.

4 Resultados experimentales

Las pruebas han sido realizadas con implementaciones de los algoritmos mencionados en el apartado anterior, parametrizando las variables W (peso máximo de la mochila) y N (cantidad de objetos).

Tabla 1. Resultados experimentales para N=20 y W=100.

Algoritmo	Tiempo (ms)	Nodos expandidos	Peso Obtenido	Valor
BackTracking	7	3713	99	2879
Las Vegas	2	361	100	2806
Voraz	1	20	95	2831

Tabla 2. Resultados experimentales para N=50 y W=100.

Algoritmo	Tiempo (ms)	Nodos expandidos	Peso Obtenido	Valor
BackTracking	258	1090920	98	4359
Las Vegas	5	1921	95	3521
Voraz	0	50	94	4255

Tabla 3. Resultados experimentales para N=100 y W=100.

Algoritmo	Tiempo (ms)	Nodos expandidos	Peso Obtenido	Valor
BackTracking	454	17119298	99	6552
Las Vegas	12	6971	100	4316
Voraz	0	60	100	6422

Tabla 4. Resultados experimentales para N=120 y W=100.

Algoritmo	Tiempo (ms)	Nodos expandidos	Peso Obtenido	Valor
BackTracking	5531	191398300	100	8220
Las Vegas	1	9494	99	5129
Voraz	0	120	98	8199

Tabla 5. Resultados experimentales para N=150 y W=100.

Algoritmo	Tiempo (ms)	Nodos expandidos	Peso Obtenido	Valor
BackTracking	60931	2007141695	99	9858
Las Vegas	1	11570	100	5387
Voraz	0	17	100	9831

Tabla 6. Resultados experimentales para N=200 y W=100.

Algoritmo	Tiempo (ms)	Nodos expandidos	Peso Obtenido	Valor
BackTracking	1219073	-1530768062	100	10839
Las Vegas	2	21534	100	5576
Voraz	1	178	100	10546

Tabla 7. Resultados experimentales para N=300 y W=100.

Algoritmo	Tiempo (ms)	Nodos expandidos	Peso Obtenido	Valor
BackTracking	119Min	-	-	-
Las Vegas	21	21016	100	6515
Voraz	1	20	100	12864

Tabla 8. Resultados experimentales para N=20 y W=200.

Algoritmo	Tiempo (ms)	Nodos expandidos	Peso Obtenido	Valor
BackTracking	37	111113	200	5019
Las Vegas	1	356	195	3577
Voraz	0	20	183	4880

Tabla 9. Resultados experimentales para N=50 y W=200.

Algoritmo	Tiempo (ms)	Nodos expandidos	Peso Obtenido	Valor
BackTracking	1934	60678669	200	7756
Las Vegas	5	1977	200	5090
Voraz	0	13	200	7756

Tabla 10. Resultados experimentales para N=100 y W=200.

Algoritmo	Tiempo (ms)	Nodos expandidos	Peso Obtenido	Valor
BackTracking	6110430	-1641880824	200	11341
Las Vegas	12	6439	200	6005
Voraz	0	69	200	11277

Tabla 11. Resultados experimentales para N=70 y W=300.

Algoritmo	Tiempo (ms)	Nodos expandidos	Peso Obtenido	Valor
BackTracking	89	269998	293	6267
Las Vegas	1	338	292	5145
Voraz	0	20	293	6111

Tabla 12. Resultados experimentales para N=120 y W=300.

Algoritmo	Tiempo (ms)	Nodos expandidos	Peso Obtenido	Valor
BackTracking	64347	1593143698	299	9401
Las Vegas	2	2145	294	6147
Voraz	0	50	295	9301

Tabla 13. Resultados experimentales para N=150 y W=300.

Algoritmo	Tiempo (ms)	Nodos expandidos	Peso Obtenido	Valor
BackTracking	209972	5265284077	300	11362
Las Vegas	3	3141	298	8385
Voraz	0	15	300	11362

Tabla 14. Resultados experimentales para N=200 y W=300.

Algoritmo	Tiempo (ms)	Nodos expandidos	Peso Obtenido	Valor
BackTracking	87Min	-	-	-
Las Vegas	6	3326	300	8039
Voraz	1	70	295	10869

Como puede observarse en las Tablas 1 al 14, el algoritmo de Bakctracking devuelve la mejor solución así como también es el que mayor tiempo de ejecución y nodos expandidos presenta. Teniendo en cuenta la Tabla 7 (N=300 y W= 100) y la Tabla 14 (N=200 y W= 300), dicho algoritmo se torna extremadamente lento, por lo que, al cabo un tiempo de ejecución considerable (119 y 87 minutos respectivamente) y al no haber arrojado resultado alguno, se decidió detener la ejecución del mismo, presentando solamente los resultados de Voraz y las Vegas, los cuales proporcionaron sus soluciones en un rango de 0ms a 21ms.

Por otra parte se puede observar al algoritmo Voraz retornar una solución muy cercana a la proporcionada por el algoritmo de Backtracking y en un tiempo que ronda entre 0ms y 1ms, presentando así una solución muy cercana a la óptima en una cantidad de tiempo despreciable.

Por su lado el algoritmo Las Vegas presenta un tiempo de ejecución mayor al Voraz y menor al de Backtracking además de soluciones no muy próximas a la soluciones brindadas por Bactracking y Voraz.

5 Conclusiones y trabajos futuros

El algoritmo de Backtracking encuentra siempre la mejor solución al problema de la mochila pero se vuelve inviable en términos de tiempo de ejecución cuando el número de objetos a ser examinados es muy elevado. En base a esto, el algoritmo puede utilizarse en aquellos casos en los que el tiempo de ejecución no esté limitado y se desee obtener la solución óptima al problema.

En casos en los que se requiera un tiempo de ejecución limitado y se acepte al menos una solución que no sea la óptima, el algoritmo voraz y las vegas son una buena alternativa. En este aspecto, según las pruebas experimentales, puede notarse que ante el problema de la mochila, el algoritmo voraz no siempre devuelve la mejor solución pero es la que más se acerca a ella y en un tiempo despreciable, en comparación con el algoritmo de las Vegas y Backtracking.

Referencias

1. Goddard, S. *Dynamic programming 0-1 Knapsack problem*. *University of Nebraska-Lincoln.* http://cse.unl.edu/~goddard/Courses/CSCE310J/Lectures/Lecture8-DynamicProgramming.pdf. Accedido en 07 de octubre de 2013.
2. Problema de la mochila. *Wikipedia.* http://es.wikipedia.org/wiki/Problema_de_la_mochila