The Knapsack problem: Optimization Approaches

Shawn Yap

School of Information Technology  
734 E North 14th St. 79601 Abilene, TX  
Apt. D  
+1(325)232-2559

Scy12a@acu.edu

Aldo Anaya

School of Information Technology

1st line of address  
2nd line of address  
Telephone number, incl. country code

axa13a@acu.edu

**ABSTRACT**

In this paper, we present several attempted algorithms to further optimize the speed of the classic Knapsack problem. The Knapsack problem is a combinatorial problem that searches for the sum of the highest values from a list of items while constrained within the cost of the knapsack is able to carry. The optimizations that we have applied on the knapsack problems are some greedy approximation algorithm which consist of four different sorts to find the *upper bound* (The highest value obtainable within the cost limit determined from the list of given set of items) and the *lower bound* (The lowest values obtainable within the cost limit determined from the list of given set of items). Besides greedy algorithms, we have also developed a state space that consists of all possible combination obtainable from the given set of items in the list. The primary goal of this paper is go explore and present a comparative study of an exhaustive search, greedy algorithm and other ways. This paper provides insights of the complexity of each algorithm in terms of time and memory.

**CCS Concepts**

• **Computing methodologies ➝Heuristic function construction;**

**General Terms**

Algorithms.

**Keywords**

Heuristic search; combinatorial optimization; the Knapsack problem;

# INTRODUCTION

The Knapsack problem is a classical problem that searches the highest combinational values from a list of items that consist of cost and values. The knapsack problem is a decision problem such that given a set of items that each consists of a cost and a value, find the highest possible value from each of the item while remaining in the range of the cost that the problem is being constrained. Which means that the cost may only remain less than or equal to the constrained cost limit. The decision form of the Knapsack problem is a NP-complete problem such that a precise solution for a huge input is nearly practically impossible to obtain.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*Conference’10*, Month 1–2, 2010, City, State, Country.

Copyright 2010 ACM 1-58113-000-0/00/0010 …$15.00.

DOI: http://dx.doi.org/10.1145/12345.67890

# The Knapsack Problem

In this section, we will define the Knapsack problem, then we will provide the details of the attempted approaches in which we have tried to get the most optimal results.

## Method Descriptions

### Greedy Solutions

The greedy approach are the primary methods that we have used to obtain the upper bound and the lower bound of the Knapsack problem. There are four type of sorts used to sort the list of items: Sort by highest-value first, sort by lowest-cost first, sort by highest ratio first, and partial knapsack. However, since it is impossible to determine if the result returned from one of the approach is the best one, all four approaches mentioned above were used to obtain the most optimal result with the cost and value being at its maximum.

#### Sort by Highest-Value First

This solution first sorts the given set of items into a collection of highest values first and then sums the values from the top of the list with the highest values while keeping the cost within the range of the limit that the problem is constrained to. Although this algorithm prioritizes the highest values in the list first, there were cases where a more optimal answer was found by looking at the items with lower values with lower cost. Thus,

#### Sort by Lowest-cost First.

This solution sorts the given list of items from the lowest cost to the highest. By doing so, we are able to keep the cost the lowest for as many iterations as possible, allowing more space for more items to be included in the knapsack. Although this method would work for a list of items with low cost and high values, this method does not produce the optimal answer when the list of items contains both low cost and low values. There were cases where a single item would have a significantly higher value than the result obtained from this greedy approach.

#### Sort by highest ratio first

This greedy approach sorts the given inputs by the highest ratio first allowing the algorithm to grab the item with the highest cost to value ratio.

#### Partial Knapsack

This greedy approach first sorts the given list of inputs by the highest cost to value ratio then grabs as many items as possible until the cost is near or equal to the limit that the problem is being constrained to. If the cost is less the constrained limit then we will continue to add part of the cost and value from the next item until we have the cost at its maximum.

### State Space Search

The state space is an approach that we have used to obtain every possible combination of the given list of items. Thus, allowing us to find the best possible combinations of the items that has its cost at its maximum and the values at its highest. However, there are issues that we have encountered from this approach: memory and time issues.

By following this approach, a given set of input items with the size of N, produces a tree with the size of . An enormous amount of memory would be consumed when N > 18. Consider a regular node in a linked list, a regular node that contained a left and a right pointer takes up about 8 bytes. If we include a string and 2 integers into the node, it all adds up to (number of character in the string \* 2 + 2 + 2 + 8) about 14 bytes per node which totals up to about (33554432 \* 14) 469762020 bytes= 0.46976202 gb when N = 24.

No calculations were included yet on what had mentioned above. Considering the function we have built that runs in O(), time is the other issue that this approach produces.

#### Alternative to Trees with Nodes in Linked-list

The approach that we have taken in alternative to build a tree which consist of linked-list nodes, we chose to build the tree using an array that consist of only ‘chars’ in each index of the array. Each index of the array represents different nodes in a visualized form of the tree. We are able to move to the left sub-tree by determining if the index is odd (left branch) or even (right branch) as shown in the code included below:



By doing so, it significantly reduced the amount of time taken to compile as compared to building a tree with nodes.

## Example

The title (Helvetica 18-point bold), authors' names (Helvetica 12-point) and affiliations (Helvetica 10-point) run across the full width of the page – one column wide. We also recommend phone number (Helvetica 10-point) and e-mail address (Helvetica 12-point). See the top of this page for three addresses. If only one address is needed, center all address text. For two addresses, use two centered tabs, and so on. For more than three authors, you may have to improvise.[[1]](#footnote-1)



Figure 1. Insert caption to place caption below figure.

.

## First Page Copyright Notice

Please leave 3.81 cm (1.5") of blank text box at the bottom of the left column of the first page for the copyright notice.

## Subsequent Pages

For pages other than the first page, start at the top of the page, and continue in double-column format. The two columns on the last page should be as close to equal length as possible.

Table . Table captions should be placed above the table

|  |  |  |  |
| --- | --- | --- | --- |
| **Graphics** | **Top** | **In-between** | **Bottom** |
| Tables | End | Last | First |
| Figures | Good | Similar | Very well |

## References and Citations

Footnotes should be Times New Roman 9-point, and justified to the full width of the column.

Use the “ACM Reference format” for references – that is, a numbered list at the end of the article, ordered alphabetically and formatted accordingly. See examples of some typical reference types, in the new “ACM Reference format”, at the end of this document. Within this template, use the style named *references* for the text. Acceptable abbreviations, for journal names, can be found here: <http://library.caltech.edu/reference/abbreviations/>. Word may try to automatically ‘underline’ hotlinks in your references, the correct style is NO underlining.

The references are also in 9 pt., but that section (see Section 7) is ragged right. References should be published materials accessible to the public. Internal technical reports may be cited only if they are easily accessible (i.e. you can give the address to obtain the report within your citation) and may be obtained by any reader. Proprietary information may not be cited. Private communications should be acknowledged, not referenced (e.g., “[Robertson, personal communication]”).

## Page Numbering, Headers and Footers

Do not include headers, footers or page numbers in your submission. These will be added when the publications are assembled.

# FIGURES/CAPTIONS

Place Tables/Figures/Images in text as close to the reference as possible (see Figure 1). It may extend across both columns to a maximum width of 17.78 cm (7”).

Captions should be Times New Roman 9-point bold. They should be numbered (e.g., “Table 1” or “Figure 2”), please note that the word for Table and Figure are spelled out. Figure’s captions should be centered beneath the image or picture, and Table captions should be centered above the table body.

# SECTIONS

The heading of a section should be in Times New Roman 12-point bold in all-capitals flush left with an additional 6-points of white space above the section head. Sections and subsequent sub- sections should be numbered and flush left. For a section head and a subsection head together (such as Section 3 and subsection 3.1), use no additional space above the subsection head.

## Subsections

The heading of subsections should be in Times New Roman 12-point bold with only the initial letters capitalized. (Note: For subsections and subsubsections, a word like *the* or *a* is not capitalized unless it is the first word of the header.)

### Subsubsections

The heading for subsubsections should be in Times New Roman 11-point italic with initial letters capitalized and 6-points of white space above the subsubsection head.

# ACKNOWLEDGMENTS

Our thanks to ACM SIGCHI for allowing us to modify templates they had developed.

# REFERENCES

1. Bowman, M., Debray, S. K., and Peterson, L. L. 1993. Reasoning about naming systems. *ACM Trans. Program. Lang. Syst.* 15, 5 (Nov. 1993), 795-825. DOI= <http://doi.acm.org/10.1145/161468.16147>.
2. Ding, W. and Marchionini, G. 1997. *A Study on Video Browsing Strategies*. Technical Report. University of Maryland at College Park.
3. Fröhlich, B. and Plate, J. 2000. The cubic mouse: a new device for three-dimensional input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (The Hague, The Netherlands, April 01 - 06, 2000). CHI '00. ACM, New York, NY, 526-531. DOI= <http://doi.acm.org/10.1145/332040.332491>.
4. Tavel, P. 2007. *Modeling and Simulation Design*. AK Peters Ltd., Natick, MA.
5. Sannella, M. J. 1994. *Constraint Satisfaction and Debugging for Interactive User Interfaces*. Doctoral Thesis. UMI Order Number: UMI Order No. GAX95-09398., University of Washington.
6. Forman, G. 2003. An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res.* 3 (Mar. 2003), 1289-1305.
7. Brown, L. D., Hua, H., and Gao, C. 2003. A widget framework for augmented interaction in SCAPE. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology* (Vancouver, Canada, November 02 - 05, 2003). UIST '03. ACM, New York, NY, 1-10. DOI= <http://doi.acm.org/10.1145/964696.964697>.
8. Yu, Y. T. and Lau, M. F. 2006. A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions. *J. Syst. Softw.* 79, 5 (May. 2006), 577-590. DOI= <http://dx.doi.org/10.1016/j.jss.2005.05.030>.
9. Spector, A. Z. 1989. Achieving application requirements. In *Distributed Systems*, S. Mullender, Ed. ACM Press Frontier Series. ACM, New York, NY, 19-33. DOI= <http://doi.acm.org/10.1145/90417.90738>.

Columns on Last Page Should Be Made As Close As Possible to Equal Length

1. If necessary, you may place some address information in a footnote, or in a named section at the end of your paper. [↑](#footnote-ref-1)