Abstraction development for the 0/1 Knapsack problem

Eli Jaffe
Mentored by Dr. Peter Schwartz

Introduction

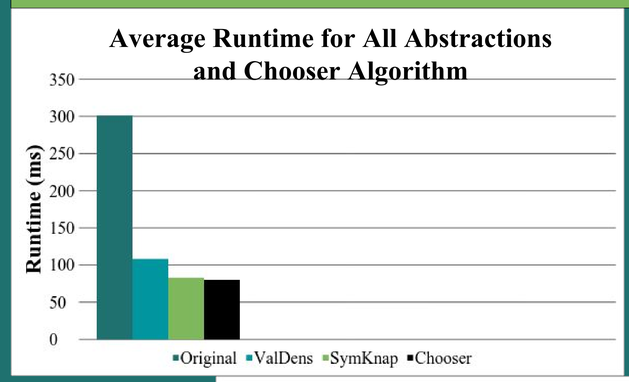
The 0/1 Knapsack is a famous problem in combinatorial optimization. The aim is to find a specific subset of a larger set of items so their summed weight is below a certain capacity and their summed value is maximized. This problem often arises in computer processor allocation. It is NP-complete, meaning its computation time grows exponentially with the problem size (Chandra, 1976). A common solution method for the 0/1 Knapsack is called the Branch and Bound. This method creates a tree of all possible item combinations, then uses a bounding algorithm to decide if a path could produce an optimal solution. However, large instances of these problems often still require abstraction. In computer science, abstraction is ignoring details of a problem to obtain an approximate solution in a more reasonable amount of time (Sahni, 1975). Development of a reliable abstraction method is vital to solving these problems in a reasonable amount of time. This investigation explored the relationships between two different abstraction methods and two different aspects of the problem instance in order to predict when these methods would be most effective.

Methods

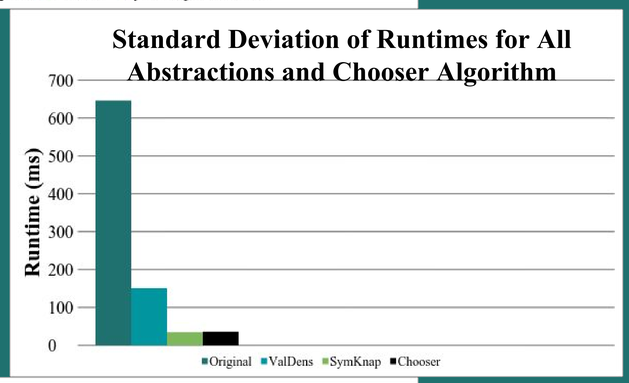
Two abstractions of the 0/1 Knapsack problem were developed, Value Density (ValDens) and Symmetrical Knapsack (SymKnap). A base algorithm for the Branch and Bound solver was taken from an online source with the permission of its owner. The ValDens and SymKnap abstractions were then coded. ValDens eliminates all items below the average value/weight ratio, also called value density. Items with low value density are generally less useful, so their elimination should minimally affect solution accuracy. Decreasing the problem size lowers the number of paths the Branch and Bound must consider, which decreases the runtime. SymKnap sorts the items by similarity based on their value densities and then alternates adding them to one of two sets. One of these sets is solved as a Knapsack with half the original capacity. For each item in the solved set, SymKnap adds corresponding items from the second set. This abstraction halves the size of the problem, which decreases runtime. Since similar items play similar roles, the final solution should be relatively accurate.

One thousand problem instances were randomly generated with max value/min value ratios between 100 and 1,000 and weight/capacity ratios between 30 and 2,500. Each instance was solved with no abstraction, ValDens, and SymKnap and each abstraction's runtime and solution were recorded for use in determining the most effective abstraction for each problem instance. A choosing algorithm was created in Excel® to return the runtime of the predicted best abstraction.

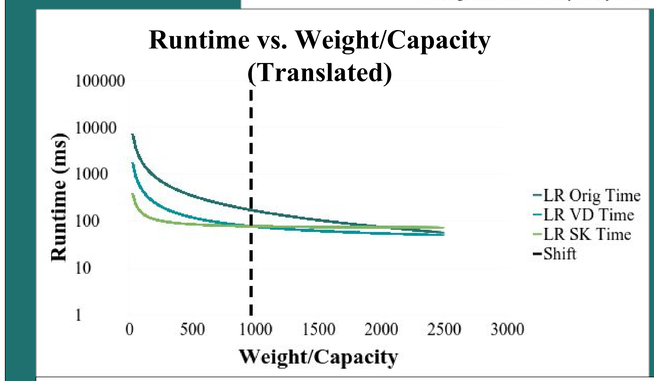
Results



Graph 1: Shows difference in average runtimes for problems solved without abstraction, with ValDens, with SymKnap, and with the choosing algorithm.



Graph 2: Shows difference in standard deviation in runtimes for problems solved without abstraction, with ValDens, with SymKnap, and with the choosing algorithm.



Graph 3: Shows the translated inverse linear relationship used to find the point where SymKnap becomes less efficient than ValDens.

Graph 1 shows a significant decrease in average runtime for each of the abstractions. On average, SymKnap lowers runtime more than ValDens. The choosing algorithm reduced average runtime slightly more than SymKnap. As seen in Graph 2, abstraction also decreased the standard deviation of runtimes. SymKnap lowered the spread much more than ValDens and slightly more than the choosing algorithm. Linear regression lines from runtime versus Capacity/Weight were translated into the inverse linear curves shown in Graph 3. The ValDens and SymKnap curves intersect when Weight/Capacity is approximately 963.53. At this point, ValDens becomes the more effective abstraction.

Conclusions

The purpose of this investigation was to identify relationships between these abstractions and any changes in runtime and/or accuracy, then use those relationships to predict the best abstraction for a given problem. Both abstractions maintained almost perfect accuracy, so the analysis focused on runtime. Graph 1 and Graph 2 show the significant increase in both efficiency and consistency with these abstractions. These increases show that both of these abstractions are effective and allows for analysis of when they are most effective. Graph 3 shows that SymKnap outperforms ValDens at lower values of Weight/Capacity. This is because ValDens removes items until total weight drops below the capacity. Otherwise, it would remove items from an already viable solution. When Weight/Capacity is high, ValDens can remove more items before reaching that point and can decrease the problem size more. However, SymKnap is relatively unaffected by Weight/Capacity ratio because it does not deal with removing items directly from the problem. Based on the shift in Graph 3,the chooser algorithm predicted which abstraction should run faster. It successfully produced an average runtime below either abstraction showing that its predictions are reliable.

It is important to acknowledge possible areas of error in this model. Despite efforts in controlling all other variables during problem solution, intricacies in how Java™ allocates memory may have skewed individual data points. In the future, more controlled hardware for testing would be preferable.

This investigation could be further developed by finding more relationships and including them in the chooser algorithm. With more relationships, different aspects of the problem instance can also be considered when predicting the best abstraction. Implementing more characteristics of the problem will increase the prediction ability of the chooser algorithm.

References

Chandra, A. K., Hirschberg, D. S., & Wong, C. K. (1976). Approximate algorithms for some generalized knapsack problems. *Theoretical Computer Science*, *3*(3), 293-304.

Sahni, S. (1975). Approximate algorithms for the 0/1 knapsack problem. *Journal of the ACM (JACM)*, *22*(1), 115-124.

Acknowledgements

First and foremost I would like to thank my classmate Kevin Merrick for his assistance in coding this project. I would also like to thank my faculty advisor Mr. Sloan who kept me organized and on schedule throughout the year.