Multi-Objective Knapsack Problem

Genetic Algorithms and Wisdom of Crowds

Sarah Mullins

Speed School of Engineering

University of Louisville

Louisville, USA

skmull02@louisville.edu

Ashley Revlett

Speed School of Engineering

University of Louisville

Louisville, USA

anrevl01@louisville.edu

**Abstract – The purpose of this project was to explore the application of genetic algorithms and wisdom of crowds to the multi-objective knapsack problem. This is a variation of the classic NP-complete knapsack problem, in which the contents of a container should have maximum value without outweighing the capacity of the container. A multi-objective version of this problem involving the maximization and minimization of given values was explored. Large values for the number of objects and knapsack capacity quickly increase the computational complexity of the problem, making brute-force techniques intractable. Instead, genetic algorithms were combined with the wisdom of crowds to attempt to find optimal solutions. The genetic algorithm converged relatively quickly toward near-optimal values, but often got stuck in local optimal solutions. Applying the wisdom of crowds approach to the results did not necessarily yield improved solutions.**

# INTRODUCTION

The original Knapsack Problem is based on the following premise: if there are *n* objects with weights *w* and values *v*, what is the optimal configuration of objects within a knapsack of capacity *c*? That is, how can *v* be maximized so that the total of all *w* is less than *c*?

Many practical applications for the Knapsack Problem can be found in an array of fields. Such examples cited by Pisinger include investors deciding which projects to fund given limited funds, a diner choosing the best meal to eat while adhering to price and calories restrictions, and loading cargo into a truck with a specific capacity for volume and weight [8].

There are several different variations of the Knapsack Problem. For example, in the 0-1 version, no object can be placed into the knapsack more than once [1]. A generalized form of 0-1 called MOKP (multi-objective Knapsack Problem) will be explored in this paper. In this type of problem, instead of having only a weight and value, other features affect the fitness of an object. In our project, each object has a price, weight, and value. The goal is to maximize the value while minimizing weight and price. This problem utilizes the concept of Pareto efficiency or optimality, which describes situations in which improving the status of one individual degrades the status of another individual [2].

MOKP is an NP-complete problem; that is, a nondeterministic polynomial one. To understand this concept, we must first define polynomial problems (P-problems). The time taken to perform a given algorithm can be described in terms of Big-Oh Notation. For example, a simple for-loop that iterates *n* times will have a time complexity of O(*n*)*.* If two of these loops were to be nested, the Big-Oh time complexity of the algorithm would be O(*n*2), and so on and so forth [4]. Polynomial problems can be solved in time O(*nk*), for some value of *k*. In nondeterministic polynomial problems (NP-problems), heuristics can be used to estimate a solution, and its accuracy confirmed in, best case scenario, time O(nk) . However, this is assuming adequate computational resources and/or lucky guessing; it is possible that the optimal solution cannot be found given the time and resources available. NP-complete problems are considered the most difficult problems within the realm of those in the NP category, and MOKP is included in this group [5].

We sought to find optimal solutions to the MOKP using both genetic algorithms (GA) and wisdom of crowds (WoC). Genetic algorithms can be used to apply the organic processes of evolution and natural selection to NP-complete problems. A population of random potential solutions for a given problem is first generated. Next, some of the best solutions are “bred” together to create new children solutions, while others are mutated or removed from the population altogether. This takes place for a given number of generations, or until a certain fitness threshold has been met [5].

WoC refers to the concept that problems can sometimes be better solved through an amalgamation of solutions given by several people, or even by the same person on several different occasions [6]. In particular, we implemented wisdom of artificial crowds (WoC), in which the solutions provided by each iteration of the genetic algorithm were combined to create a hopefully optimal solution [7].

Coello describes the application of genetic algorithms to multiobjective optimization (EMOO) in his 2001 article. He defines EMOO as a conglomeration of decision variables that satisfy constraints and optimize the fitness of a function. He outlines methods for addressing this issue, and we decided to use aggregation, in which the objectives are combined into one function [3].

# Literature Review

# Approach

When *n* and *c* have large values, the number of possible knapsack “packings” becomes exponentially large, and testing each for an optimal solution becomes intractable. As an alternative, our approach used GA to explore this search space for a near-optimal solution in a more computationally efficient manner. Because GA algorithms can get stuck in local maxima and miss the global optimum, it is run for several trials, and all the solutions are considered using WoC approach. The program structure and algorithms work like so:

1. Build an initial population of random solutions, encoding the solution’s items into a “chromosome”-like list of 1’s and 0’s.
2. Evolve the population over many generations by mutating and crossing over the chromosomes (more information below). The fitness of a solution increases with the value of the boxes contained in the solution, and decreases with the price and weight. To assess fitness, normalize the scores for each objective, then aggregate them. Return the best solution found during the evolutionary process.
3. Repeat steps a-b to create each member of the crowd.
4. Try to create a “wiser” solution by selecting items common to most of the crowd’s solutions, then adding any other items that will fit.
5. Return the best solution found.

For step b), the following crossovers and mutations were performed:

1) Remove chromosomes exceeding the knapsack capacity  
2) Clone the chromosomes with the highest value

3) Breed children using crossover until the population reaches its original size. Crossover involved combining the first half of the first parent with the second half of the second parent. If the result was over capacity, the items with the lowest fitness levels were then removed.

4) Mutate random children. Mutation involved removing or adding a random number of boxes at   
random chromosome indices.

# IV: Experimental Results

*A: Data*

We created data structures for the virtual knapsack and the items to be placed in it. The knapsack was given an arbitrary capacity, and each item random values for multiple objectives: weight, value and price. Each objective was then normalized (e.g. weight divided by maximum possible weight). In the case of objectives that we want to minimize (weight and price), the scores were inverted; since higher weight values are less desirable, for example, their score needed to be relatively low. The total fitness was found by summing the weight, value and price scores.

To represent each “chromosome,” random configurations of items within the knapsack were encoded with 0s and 1s. A 1 value indicates that an item is inside the knapsack, while a 0 value indicates that it is not.

*B: Results*

The figures below depict the convergence of the GA solutions over 70 generations, with population size of 60 and crowd size of 25. The variable *n* represents the number of candidate items, while *c* represents the capacity of the knapsack. As both *n* and *c* increase, the GA solutions converge more quickly over multiple generations, and the amount of diversity is reduced.

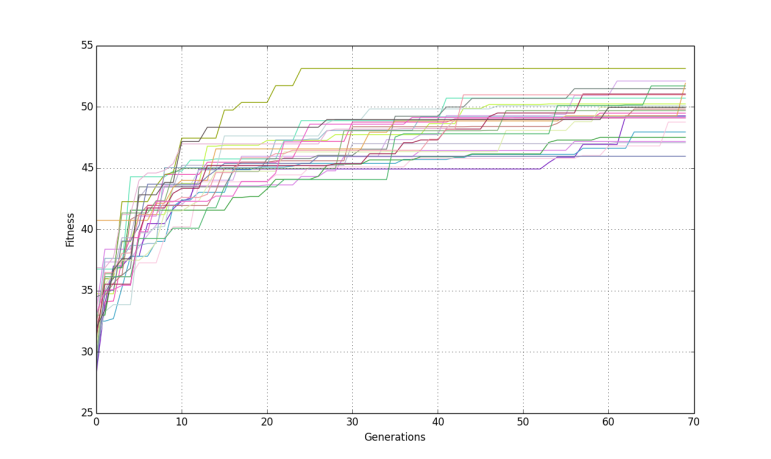
****

Figure 1: Visualization for n = 100, c = 300. Run time: 15.34s

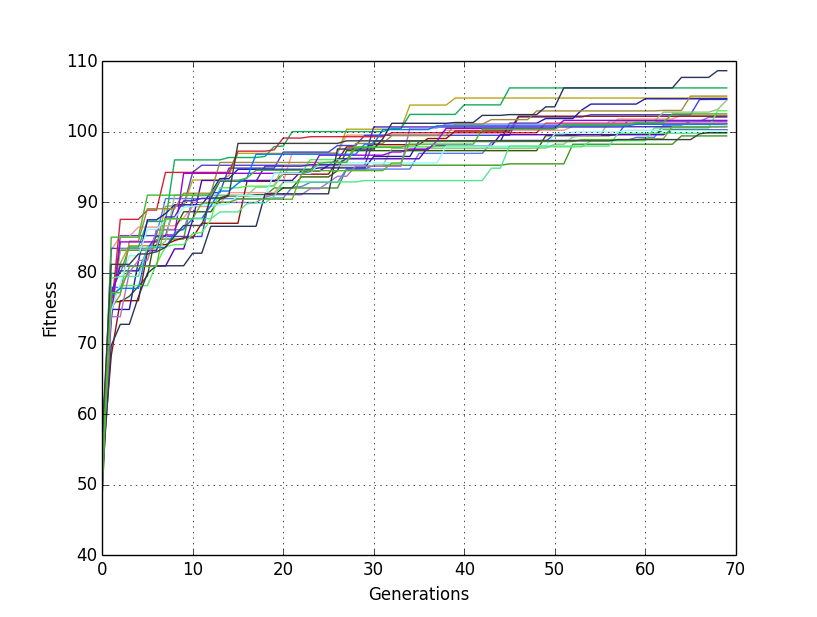
****

Figure 2: Visualization for n = 300, c = 600. Run time: 90.55s

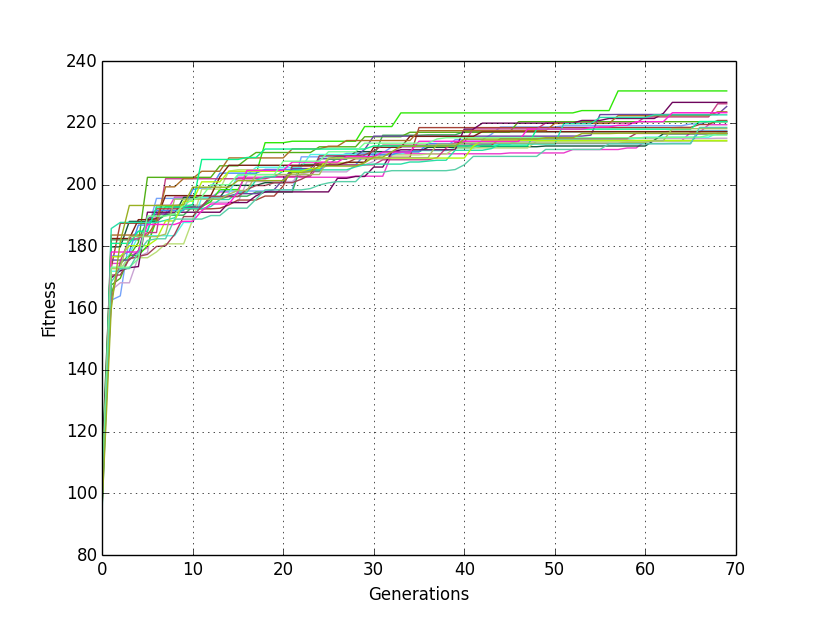
****

Figure 3: Visualization for n = 600, c = 1200. Run time: 316.61s

Overall, WoC did not produce better results than those produced by individual GA trials; they generally provided solutions with the same fitness levels, or sometimes, even worse.

|  |  |  |  |
| --- | --- | --- | --- |
| **c** | **n** | **Best GA Fitness** | **WoC Fitness** |
| 200 | 10 | 10.44 | 4.77 |
| 200 | 20 | 18.98 | 8.01 |
| 200 | 40 | 21.16 | 7.89 |
| 200 | 80 | 39.43 | 13.69 |
| 200 | 160 | 53.98 | 13.97 |
| 200 | 320 | 106.85 | 38.68 |
| 200 | 1280 | 87.56 | 23.08 |

Table 1: Comparison between GA and WoC solutions

As the values *n* (number of items) increased, so too did the run-time. The figure below depicts this exponential growth.

Figure 4: Runtime increasing with number of items

# V: Conclusions

In our project, WoC performed at about the same level of GA. However, in future work, we hope to improve the crowd-sourcing algorithm and produce results better than those achieved by GA. Another improvement that could be made is adding weights to each objective according to their importance. For example, if a low price is of more importance to a shopper than a heavy knapsack, its value would be multiplied by some factor greater than 1 and/or the weight component multiplied by a factor less than 1.

Increasing the number of objectives would also be an interesting variation of this problem. In our food example, we ultimately decided to combine all nutritional information into a single value. However, in future research, this could be divided into separate components such as protein, fat, carbohydrates, etc. As a result, calculating the fitness level of an item configuration would grow more complex. This is reminiscent of the evaluation functions of some artificial intelligence chess players, which are weighted by particular features of the board [9].

While WoC did not prove to be particularly effective in our work, GA did successfully converge to optimal solutions. Improving the parameters of the project could very well reveal the effectiveness of WoC.

# VI: Acknowledgments

The authors would like to thank Dr. Roman Yampolskiy for sharing his in-depth knowledge of GA and WoC, as well as his support and guidance in the writing of this paper.

# VII: References

[1] M. Hristakeva and D. Shrestha, "Solving the 0-1 Knapsack Problem with Genetic Algorithms," Simpson College, Midwest Instruction and Computing Symposium. 2004.

[2] C. L. Mumford, "Comparing Representations and Recombination Operators for the Multi-Objective 0/1 Knapsack Problem," *Evolutionary Computation,* vol. 2, pp. 854-861, 8-12 Dec. 2003.

[3] C. A. C. Coello, "A Short Tutorial on Evolutionary Multiobjective Optimization," in *Evolutionary Multi-Criterion Optimization*. vol. 1993, ed: Springer Berlin Heidelberg, 2001, pp 21-40.

[4] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed.: Pearson Education, Inc., 2010.

[5] M. A. Weiss, *Data Structures and Algorithm Analysis in C++*, 3rd ed.: Pearson Education, Inc., 2006.

[6] S. K. M. Yi, M. Steyvers, M. D. Lee, and M. J. Dry, "Wisdom of the Crowds in Traveling Salesman Problems”

[7] R. Yampolskiy, L. Ashby and L. Hassan, "Wisdom of Artificial Crowds—A Metaheuristic Algorithm for Optimization," *Journal of Intelligent Learning Systems and Applications*, Vol. 4 No. 2, 2012, pp. 98-107.

[8] D. Pisinger, "Algorithms for Knapsack Problems," Ph.D., Dept. of Computer Science, University of Copenhagen, Copenhagen, Denmark, 1995.

[9] R. Yampolskiy. CECS 545. Class Lecture, Topic: “Adversarial Search.” University of Louisville, Louisville, USA. Sept. 24, 2014.