**Project 6: Knapsack Problem and Genetic Algorithms/Wisdom of Crowds**

Sarah Mullins and Ashley Revlett  
Computer Engineering and Computer Science

Speed School of Engineering

University of Louisville, USA

skmull02@louisville.edu, anrevl01@louisville.edu

1. **Introduction**

In this project, we investigated the Knapsack Problem, an NP-complete combinatorial optimization problem. The premise behind this problem is maximizing the value *v* of *n* discrete objects with weights *w* within a knapsack without exceeding its capacity *c*. Specifically, we used a derivation of the problem called “0-1,” meaning that only one instance of an object can appear in the knapsack [1]. This version of the problem was used in a paper by M. Hristakeva, D. Shrestha, and was the basis for our project. One strategy for finding the best combination of objects is the so called “greedy approximation algorithm,” in which ratios of value to weight for each object are determined and used in ascending order [3]. However, we combined genetic algorithms with the concept of wisdom of crowds to attempt to find optimal solutions for randomly created Knapsack Problems.

1. **Approach**

When *n* and *c* have large values, the number of possible knapsack “packings” becomes exponentially large. Our approach was based on using a genetic algorithm to explore this search space for a near-optimal solution in a more computationally efficient manner by using a genetic algorithm (GA). Because genetic algorithms can get stuck in local maxima and miss the global optimum, the GA is run repeatedly, and all GA solutions are considered using a “wisdom of crowds” approach. The program structure and algorithms work like so:

* 1. Build an initial population of random solutions, encoding the solution’s boxes 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, so long as the total weight of the boxes does not exceed the knapsack’s capacity. 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 boxes common to most of the crowd’s solutions, then adding any other boxes 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  
 Note: Crossover involved combining the first half of the first parent with the   
 second half of the second parent.

4) Mutate children with the lowest value  
 Note: Mutation involved removing or adding a random number of boxes at   
 random chromosome indices.

1. **Results** 
   1. **Data**  
      All results shown below use the following parameters:

*n* = 12

*c* = 100

Domain(*w*) = range(1, 50)

Domain(*v*) = range(1, 50)

Crowd size = 20

Population size = 20

Generation = 30

Mutation rate = 10%

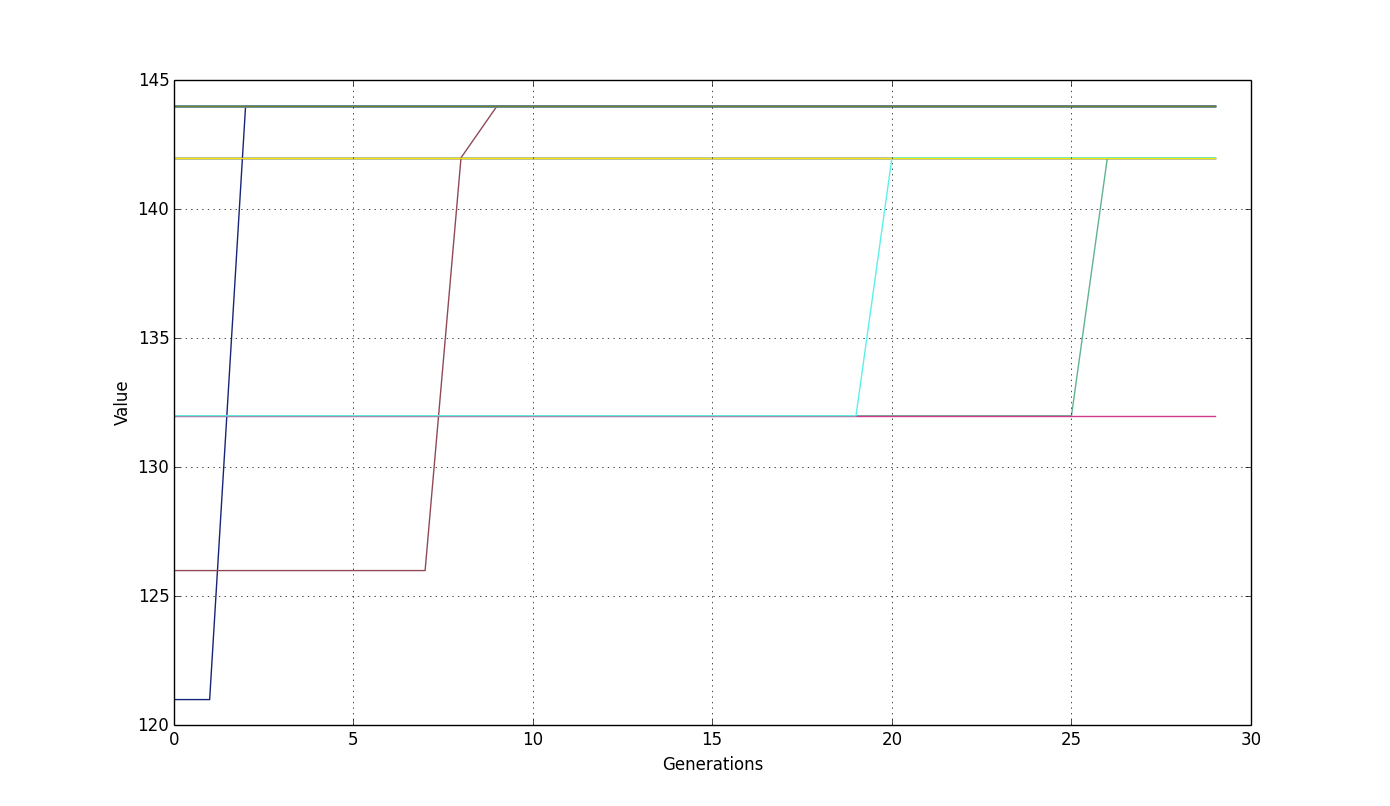
Parent cloning rate = 40%

Specific weights and values for each box are provided in the appendix.

While the domain of values was arbitrary, the domain of weights, number of objects and knapsack capacity are all strongly interconnected. With the parameters given here, there was a small possibility that all objects could fit inside the knapsack if their randomly chosen weights were all 8 units or less. However, we felt this result was relatively improbable, and if the average weight is indeed around 25 units, there are an adequate number of possible object combinations to consider.

Each crowd member uses a genetic algorithm to evolve a solution over a number of generations. The population pool used in that evolution gradually converges on a solution. The chart below shows the improvements made to the best solution found so far over 30 generations. Each color line is a different crowd member. Within the first 20 generations, mutations and crossovers produce significant jumps in the value of the best-known solution. By generation 30, most crowd solutions converge toward a near-optimal value, but some get stuck in a local optima. These results are shown in Figure 1.

*Figure 1. Performance of Genetic Algorithm for n=12 Over 30 Generations*

****

*Table 1. Results of GA Evolution and WoC Optimization*

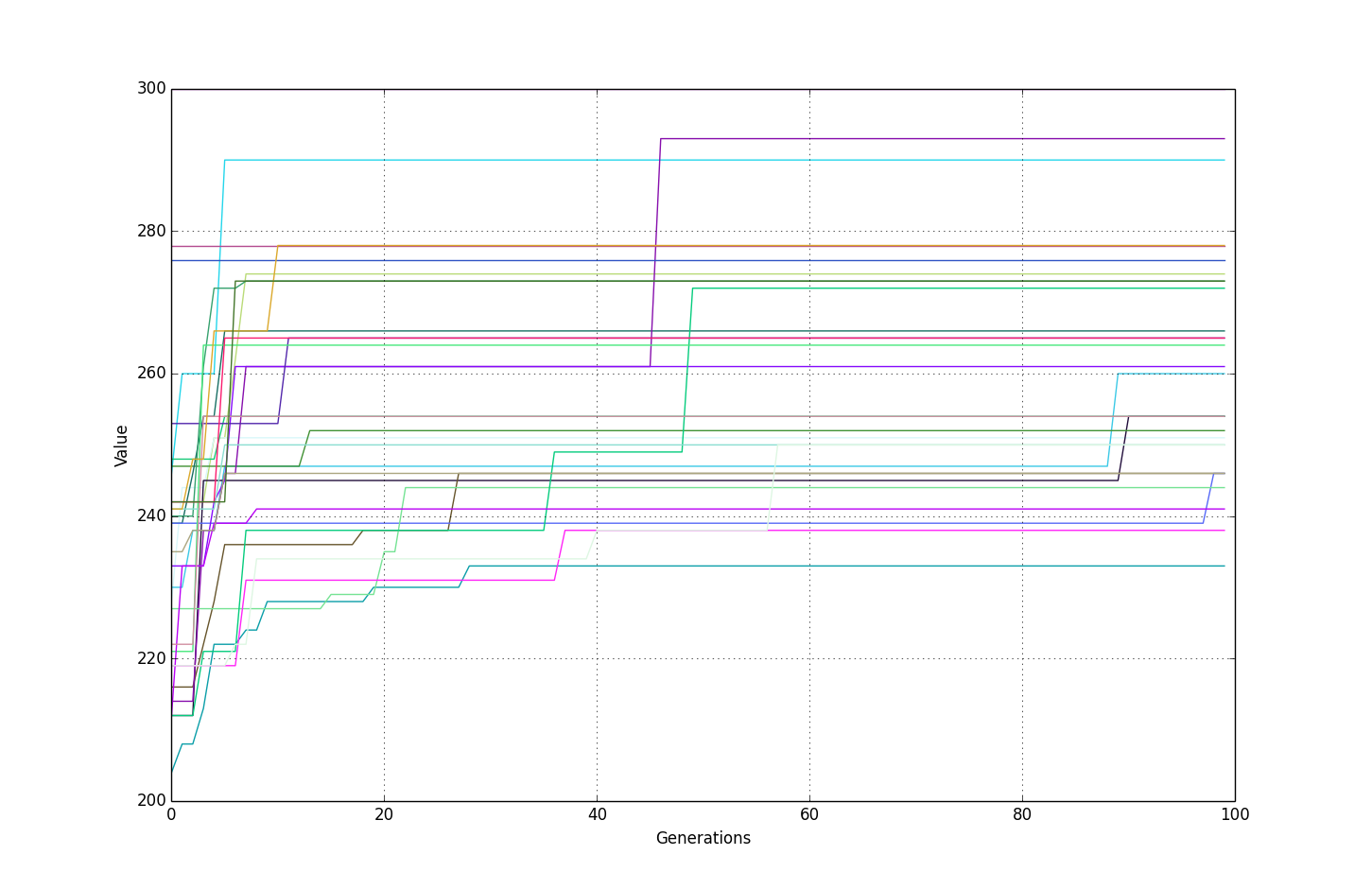
|  |  |  |  |
| --- | --- | --- | --- |
| Crowd Member # | Solution | Value | Weight |
| 0 | [1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0] | 142 | 50 |
| 1 | [1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0] | 144 | 45 |
| 2 | [1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0] | 142 | 50 |
| 3 | [1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0] | 144 | 45 |
| 4 | [1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0] | 142 | 50 |
| 5 | [1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0] | 144 | 45 |
| 6 | [1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0] | 144 | 45 |
| 7 | [1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0] | 144 | 45 |
| 8 | [0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0] | 132 | 50 |
| 9 | [1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0] | 144 | 45 |
| 10 | [1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0] | 144 | 45 |
| 11 | [1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0] | 142 | 50 |
| 12 | [1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0] | 144 | 45 |
| 13 | [1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0] | 142 | 38 |
| 14 | [1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0] | 142 | 50 |
| 15 | [1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0] | 144 | 45 |
| 16 | [1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0] | 144 | 45 |
| 17 | [1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0] | 144 | 45 |
| 18 | [1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0] | 144 | 45 |
| 19 | [1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0] | 144 | 45 |

|  |  |  |  |
| --- | --- | --- | --- |
| WoC Solution | Solution | Value | Weight |
| 0 | [1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0] | 144 | 45 |

The Wisdom of Crowds optimization strategy also discovered the same near-optimal solution as most of the individual crowd members. It could not find any improvements to that solution.

A larger problem with parameters of *n* = 50 and *c* = 300 was also tested. In this test, the crowd size was 30, population size was 50, and number of generations was 100. As shown in Figure 2, larger problems exhibit the same behavior, but with greater diversity. Many solutions approach the optimal packing, but many also get stuck in a mid-range local optima.

*Figure 2. Performance of Genetic Algorithm for n=50 Over 100 Generations*

****

Using the Wisdom of Crowds approach, we try to construct an even better solution by identifying which boxes are shared by most solutions, then iterating further on that solution. Even if we are unable to create a better solution through this merging, we can still use the best solution provided by the GA.

* 1. **Results**

*Table 2. Runtimes for various n and c*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Runtime | n | c | Crowd Size | Population Size | Generations |
| 1s 253ms | 12 | 100 | 20 | 15 | 30 |
| 1s 124ms | 50 | 200 | 20 | 15 | 30 |
| 3s 330ms | 50 | 300 | 20 | 50 | 50 |
| 14s 178ms | 100 | 500 | 30 | 50 | 50 |
| 46s 965ms | 200 | 1000 | 30 | 50 | 50 |
| 2m 49s 690ms | 400 | 1000 | 30 | 50 | 50 |

*Figure 2. Runtimes for Table 2*

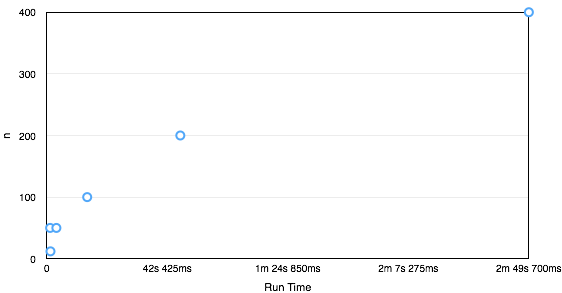
**

Table 2 and Figure 2 show the runtimes for a variety of configurations. As *n* and *c* increase, the runtime of the program grows at a rate that is less than the exponential growth a brute-force approach would cause.

1. **Discussion**

The Knapsack problem is especially well-suited to a genetic approach because the solutions are simple to encode into chromosome-like lists. These lists are the same length as the number of boxes. Each place in the list contains a 0 or 1, depending upon whether the box with the ID value that matches that index is included in the collection. Using this encoding, it’s simple to mutate and cross-over chromosomes.

Using the Wisdom of Crowds permits us to use the best results from repeated runs of the genetic algorithm. Although each GA solution begins with a randomly created population, it quickly converges on a single solution. Because of this, GAs can become “stuck” in local optima, and unless very lucky will not discover the global optima. By building a population of solutions generated by GAs, we increase the diversity of our solutions and improve the chances of finding the global optima.

GAs and WoC work together to provide an ideal strategy for solving the Knapsack problem. The GA is able to efficiently explore promising search paths in a exponential search space, while the WoC distills the GA solutions into a consensus solution.

For our presentation and research paper, we plan on addressing the multi-objective knapsack problem. In contrast to the pure “0-1” version, in which a single value is combined with a single weight for each object, the multi-objective version considers several different other factors when maximizing value. Depending on the nature of the factor, each one must be minimized or maximized in order to produce the highest overall value. Due to project oversights and time constraints, we did not integrate multiple objectives into this project, but we will immediately explore this option.

1. **References**

[1] M. Hristakeva, D. Shrestha. "[Solving the 0-1 knapsack problem with genetic algorithms](http://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0CCAQFjAA&url=http%3A%2F%2Fwww.sc.ehu.es%2Fccwbayes%2Fdocencia%2Fkzmm%2Ffiles%2FAG-knapsack.pdf&ei=i1BpVPaUI8H-yQS4ioHAAg&usg=AFQjCNGHSFIxIOKD6mOxSxL55NW7tYN_Qw&sig2=-cTGRGSW7vbnVl6x-D3PFw)." *Midwest Instruction and Computing Symposium*. 2004.

[2] [Richard M. Karp](https://en.wikipedia.org/wiki/Richard_M._Karp) (1972). ["Reducibility Among Combinatorial Problems"](http://cgi.di.uoa.gr/~sgk/teaching/grad/handouts/karp.pdf). In R. E. Miller and J. W. Thatcher (editors). *Complexity of Computer Computations*. New York: Plenum. pp. 85–103.

[3] [George B. Dantzig](http://en.wikipedia.org/wiki/George_Dantzig), Discrete-Variable Extremum Problems, Operations Research Vol. 5, No. 2, April 1957, pp. 266–288,

1. **Appendix**

*Table 2. Values of Boxes used in 12-Box Experiments*

|  |  |  |
| --- | --- | --- |
| ID | Value | Weight |
| 0 | 35 | 7 |
| 1 | 16 | 37 |
| 2 | 29 | 44 |
| 3 | 25 | 19 |
| 4 | 25 | 7 |
| 5 | 43 | 20 |
| 6 | 39 | 4 |
| 7 | 35 | 38 |
| 8 | 3 | 45 |
| 9 | 2 | 7 |
| 10 | 22 | 31 |
| 11 | 14 | 15 |