Improving Genetic Algorithm for 0 - 1 Knapsack Problem

Sanmesh S Bhosale
Computer Science Department
San Jose State University
San Jose, CA 95192
+1 (669) 260-3069
sanmesh.bhosale@sjsu.edu

ABSTRACT

The 0 - 1 knapsack problem is a combinatorial optimization problem which maximizes the value of potential items to be placed in the knapsack without exceeding its size constraints. It has many proposed solutions like greedy search, brute force solution, dynamic programming, branch and bound, and backtracking. The main challenge with 0 - 1 knapsack problem is solving it in polynomial time. The Genetic Algorithm which is inspired from Darwin's Theory of evolution has proven to be successful for solving other similar NP problems like the Travelling Salesman Problem. The basic Genetic Algorithm can be modified or tuned to produce an improved genetic approach which will solve the 0 - 1 knapsack problem in polynomial time. A comparative study is performed on David Pisinger's optimization codes to evaluate the performance of an improved Genetic Algorithm against other popular solutions. Some of the initial results show that the improved Genetic Algorithm outperforms other popular solutions in terms of time complexity and provides an approximate solution which is almost as good as the best solution.

Keywords – Knapsack Problem, Genetic Algorithm (GA), Encoding, Fitness, Selection, Crossover, Mutation.

1. INTRODUCTION

A Non-Polynomial (NP) problem is defined such that it has no algorithmic solution that solves it in polynomial time. Although an algorithm can make a guess and check if it is the optimum value in the polynomial amount of time. Some examples of the NP problems are Travelling Salesman Problem, Dominating Set Problem, Clique Problem, Hamiltonian Path Problem, and Knapsack Problem [7]. The 0 - 1 knapsack problem can be solved with greedy search or dynamic programming but, a solution by these problems makes it an NP-complete problem. The Brute Force solution to the 0 - 1 knapsack problem is achieved in 2^n i.e. exponential time. Since the knapsack problem has many real-world applications, we need a solution that is achieved in polynomial time.

This paper proposes and implements an improved Genetic Algorithm which gives an approximate solution to the 0 - 1 knapsack problem in polynomial time. Even though no algorithm can guarantee the best solution to the NP problem in polynomial time the Genetic Algorithm can guess an optimal solution and check it in polynomial time.

Genetic Algorithm is randomized in nature and derives influence from the biological process which follows the law of natural selection and genetics. It generates a random population of chromosomes which are evaluated after each generation to check their fitness [4]. Once the termination condition is met the algorithm provides an approximate solution in polynomial time to beat the time complexities of greedy search, dynamic programming, and brute force solution.

2. 0 - 1 KNAPSACK PROBLEM

A. Problem Description

The knapsack problem is a very popular problem in the field of computer science. It has been well studied and researched with a number of solutions that are readily available. The problem is no algorithm can guarantee the best solution to the problem in polynomial time. That makes it an NP-hard problem with many practical applications like budgetary control, cargo loading problem, investment portfolio selection, and resource allocation problem [5].

The 0 - 1 Knapsack problem is a combinatorial optimization problem in which there exists a knapsack of a predetermined fixed capacity C, and list of potential items with weight w and value v, where the subject must maximize the value of potential items to be placed in the knapsack without exceeding its size constraints [5].

B. Mathematical Model

The 0 - 1 knapsack problem can be mathematically represented as:

$$Maximize \quad \sum_{i=1}^{n} x_i v_i$$

$$While, \quad \sum_{i=1}^{n} x_i w_i \leq C$$

$$Where, \quad x_i \begin{cases} 1 & if\ item\ in\ the\ bag \\ 0 & if\ item\ not\ in\ bag \end{cases}$$

The aim is to find a chromosome which fills the knapsack to its optimum capacity in the shortest possible time. This chromosome will have the highest fitness value in the improved Genetic Algorithm and will be a product of the last generation.

3. GENETIC ALGORITHM

Genetic Algorithm was developed by John Holland at the University of Michigan in the 1960s. Since its inception, it has found a wide variety of real-life applications and it has been found useful for particularly solving NP problems. It is a search technique used in computing to find an approximate solution to optimization problems. Since it only provides an approximate solution it is not the best available solution, but the closest optimum solution given in the polynomial amount of time. It is considered easy to implement and has fast processing times compared to other algorithms. This works as an added advantage when even an approximate solution to the problem is acceptable.

Genetic Algorithms are a class of evolutionary algorithms that use techniques inspired by evolutionary biologies such as inheritance, mutation, selection, and crossover. It mimics the process of natural evolution to create new generations or solutions in the population until a termination condition is met. Since it has randomization, it also introduces diversity in the population and is capable of finding a unique solution. The randomization of the Genetic Algorithm also makes sure that it does not get stuck at a local optimum solution and achieves a globally optimum solution.

The algorithm begins with an initial population of randomly generated strings of 0's and 1's called the Chromosomes [3]. Each 0 and 1 in this chromosome is called a Gene. The value of Gene is determined based on if the item is included in the knapsack or not. Once initial random populations are generated their fitness is evaluated by a Fitness function. Then the Selection process selects the Chromosomes according to their fitness values for further genetic operations [6]. First, the Crossover operation will take the two selected Chromosomes or Parents and selects a Crossover point to breed them and produce the Offspring. The Offspring then goes through Mutation operation where it is decided on a certain probability if the particular Gene inside the Chromosome will mutate or not. Once all the genetic operations are completed the current population is replaced by the new population and the Termination condition is evaluated. If the condition is met the algorithm stops and the generated optimal solution is provided, else it loops all the operations from fitness to produce more and more generations [3].

A. Pseudo Code of Genetic Algorithm

1. Start: Randomly generate a population of N chromosomes.
2. Fitness: Calculate the fitness of all chromosomes.
3. Selection: According to the selection method select 2 chromosomes from the population.
4. Crossover: Perform crossover on the 2 selected chromosomes.
5. Mutation: Perform mutation on the obtained chromosomes.
6. Replace: Replace the current population with the new population.
7. Test: Test whether the end condition is satisfied. If yes, stop. If no, continue.

B. System Architecture of improved Genetic Algorithm

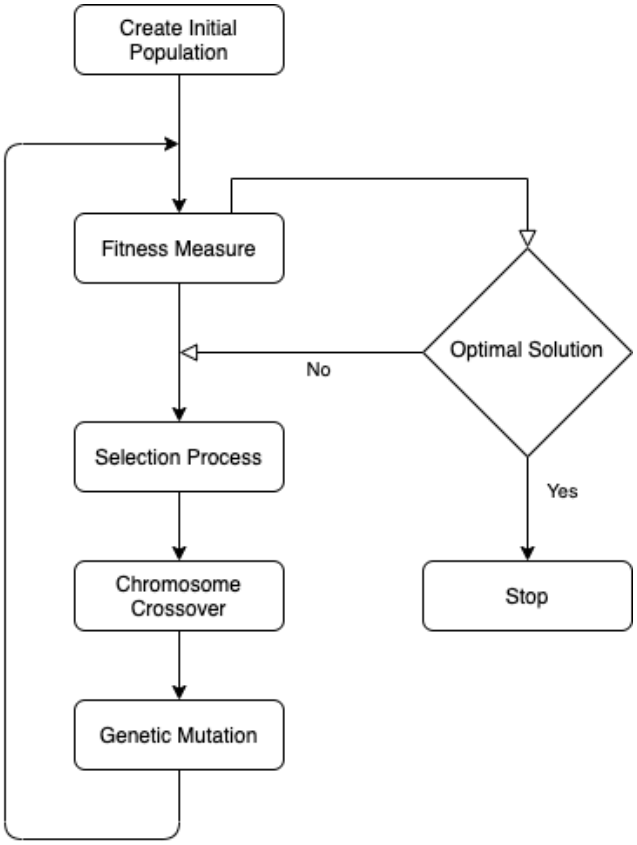


Figure 1. Flow of improved Genetic Algorithm.

1) Chromosome:

A Chromosome is represented as a string of parameters. The Chromosome for this Genetic Algorithm is represented by a string of 0's and 1's and is generated randomly at the initiation process. These multiple strings of randomly generated Chromosomes together form the initial population of our algorithm. In the improved GA implemented by this paper, the initial population is twice the number of items if the number of items is less than 32, else it is the same as the number of items. A single element in the string of chromosome is known as the Gene.

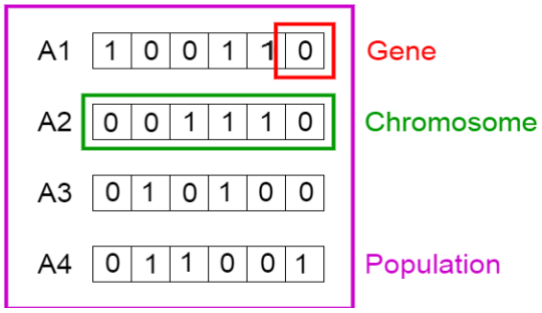


Figure 2. Gene, Chromosome, and Population structure.

2) Fitness function:

Fitness function is used to evaluate if the given solution is optimal or not. The Fitness function will sum all the values of all items that are added to the knapsack and will penalize the Chromosome for each item that increases the weight over the given constraint. The higher the fitness of a Chromosome the higher its chances of getting selected to the next generation. Fitness function ranks the Chromosomes according to their fitness value for the selection process.

3) Selection:

The selection process is very crucial for the performance of the algorithm as it decides the Chromosomes or Parents that will go under the genetic operations. A good selection process can select better Chromosomes and introduce the right amount of diversity to the algorithm. A commonly used selection process is the Roulette Wheel selection which provides each Chromosome with some region on the wheel based on its fitness values. The wheel then spins and the Chromosome that lands in front of the fixed point is selected for further genetic operations.

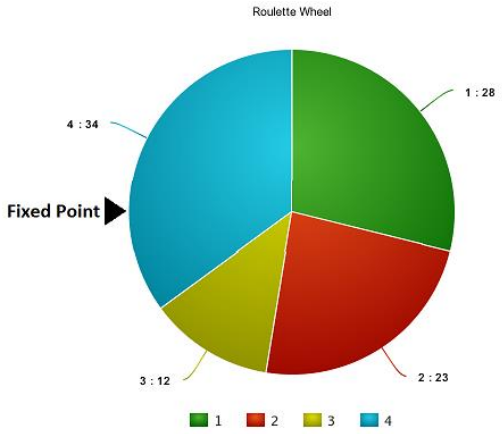


Figure 3. Roulette Wheel selection.

4) Crossover:

Crossover is the first genetic operation which takes the selected Chromosomes or Parents as input and produces an Offspring. The crossover point can be selected at random and there can be multiple crossover points. The improved GA implemented in this paper selects the crossover point at the middle or halfway of the Chromosome and the crossover rate is conservatively set to 0.8 value. It introduces diversity in our new population which will help the algorithm to produce more fit chromosomes in the coming generations.

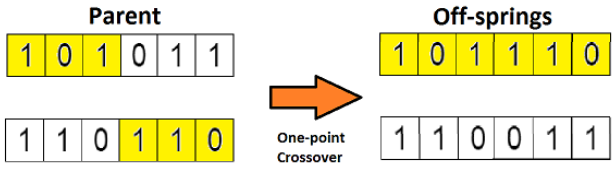


Figure 4. Midpoint Crossover.

5) Mutation:

Mutation is defined as the process of introducing of a random tweak in the Chromosome, which promotes the idea of diversity in the population. It depends on a certain randomly generated probability if a particular Gene in the Chromosome will undergo mutation or not. The improved GA implemented by this paper again conservatively selects a mutation rate of 0.5 value. Mutation makes sure that the algorithm does not get stuck at a local optimum.

Figure 5. Mutation.

6) Termination:

Setting the right termination criteria can be very tricky for the algorithm as we need to maintain a balance between execution time and optimal solution. Some examples of termination condition are: terminate when more than 70% population has the same fitness, a certain number of generations have evolved, and population fitness has not changed for n generations. The improved GA implemented by this paper terminates after 50 generations if the number of items to choose from are less than 100 and terminates after 100 generations if the number of items is more than 100.

4. HISTORY AND BACKGROUND

D. Pisinger in his 1995 thesis, “Algorithms for Knapsack Problems” has explained various algorithmic approaches to the NP-hard Knapsack problems in detail [1]. Since then various researches have focused on solving the Knapsack problem with different algorithmic approaches. As the Knapsack problem is NP problem the algorithmic approaches like dynamic programming, backtracking, and branch & bound are not the best for solving it in polynomial time. J. Holland proposed and developed the Genetic Algorithm in the 1960s which has been an effective solution to solve NP problems like the knapsack problem. T. Pradhan has studied a coalesced approach to solving the 0 - 1 knapsack problem by combining the Genetic Algorithm and Rough Set Theory [2]. More research on the topic can find an improved Genetic Algorithm that solves the 0 - 1 knapsack problem in polynomial time.

5. OTHER ALGORITHMS

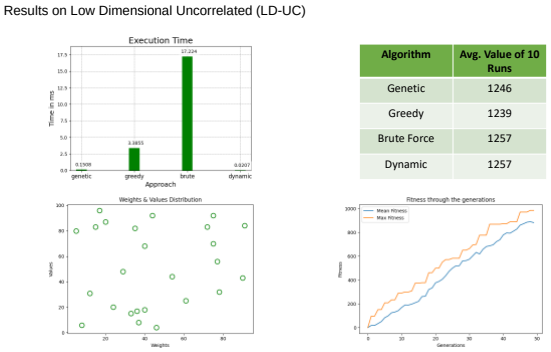
There are some other algorithms that have been implemented to perform a comparative study of their time complexities and performance against the improved Genetic Algorithm. Other algorithms implemented in this paper are greedy search, brute force, and dynamic programming. Brute force produces a solution to the problem in 2^n i.e. exponential time. Greedy search and dynamic programming perform much better in terms of time complexity and performance compared to brute force but a solution in polynomial time is not guaranteed. The paper further discusses the performance of all these algorithms against the improved GA.

6. DATASET

The dataset is obtained from David Pisinger's optimization codes [1]. The dataset is unique and is comprised of four different categories. The categories can be listed as follows: Low Dimensional Uncorrelated (LD-UC), High Dimensional Uncorrelated (HD-UC), High Dimensional Weakly Correlated (HD-WC), and High Dimensional Strongly Correlated (HD-SC). The dataset has a high variation in the dimension of values that we feed as input. This covers knapsack problems with low-value inputs as well as high-value inputs. It also contains uncorrelated, weakly correlated, and highly correlated data points which help us to understand their influence on the performance of the improved GA.

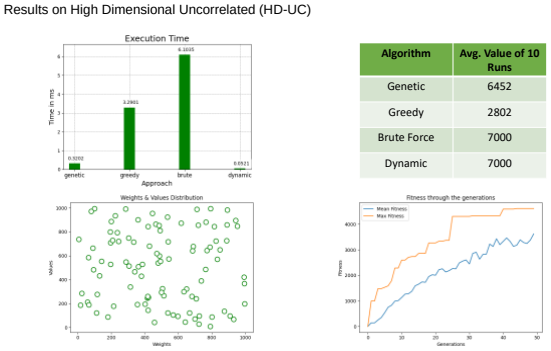
7. EXPERIMENTAL RESULTS

The experiments were performed on an Intel Core i9 (8 cores) processor with 16 GB DDR4 RAM. They were implemented in Python coding language and any Python compiler 3.7 and above will be able to run the experiments. The time complexities and performance of all other implemented algorithms against improved GA on all the four different datasets are shown in the figures that follow. The LD-UC dataset has only 25 data points while all the HD datasets have 100 data points each. For experimental purposes, the weight capacity of knapsack was fixed to 1000. The windows in each image below show the execution time of each algorithm, knapsack value that was generated, weights and values distribution on that dataset, and the fitness value of the Genetic Algorithm through each generation.



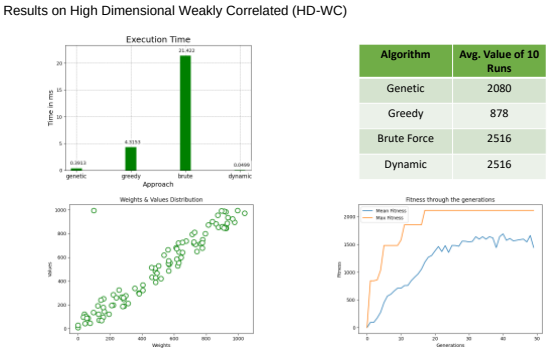
Algorithm	Avg. Value of 10 Runs
Genetic	1246
Greedy	1239
Brute Force	1257
Dynamic	1257

Figure 6. Results on Low Dimensional Uncorrelated data.



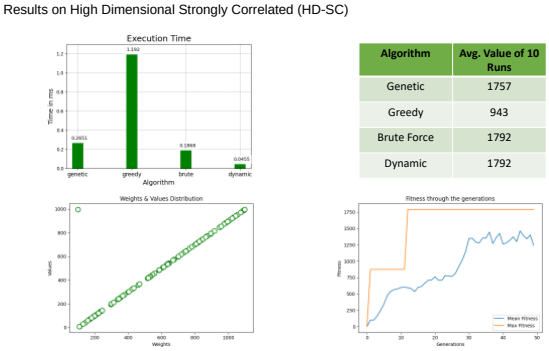
Algorithm	Avg. Value of 10 Runs
Genetic	6452
Greedy	2802
Brute Force	7000
Dynamic	7000

Figure 7. Results on High Dimensional Uncorrelated data.



Algorithm	Avg. Value of 10 Runs
Genetic	2080
Greedy	878
Brute Force	2516
Dynamic	2516

Figure 8. Results on High Dim. Weakly Correlated data.



Algorithm	Avg. Value of 10 Runs
Genetic	1757
Greedy	943
Brute Force	1792
Dynamic	1792

Figure 9. Results on High Dim. Strongly Correlated data.

8. APPLICATIONS

The Genetic Algorithm has found a lot of real-life applications for itself due to its inspiration from evolutionary biology. It has been extensively used to solve NP problems like Knapsack Problem and Travelling Salesman Problem. It is used in Robotics for trajectory planning. It also works for budgetary control and investment portfolio selection [5]. And it is also extensively used in machine learning for designing neural networks and improving classification.

9. CONCLUSION

Genetic Algorithms can be used to find an approximate solution to the 0 - 1 knapsack problem. It manages to reduce the time complexity of the 0 - 1 knapsack problem from exponential to polynomial time. The overall time complexity for GA is lower than Greedy Search and Brute Force but Dynamic Programming is always faster.

10. FUTURE WORK

Other algorithmic solutions to the 0 - 1 knapsack problem like Branch and Bound and Backtracking can also be implemented and evaluated against the improved GA. An arithmetic Crossover function which uses the logical AND or logical OR can be implemented to generate new Offspring. The concept of Elitism in which the fittest Chromosomes are directly copied in the new population can also be tested to improve the performance of the Genetic Algorithm.

11. REFERENCES

[1] D. Pisinger, “Algorithms for Knapsack Problems”, *PhD Thesis, University of Copenhagen, Dept. of Computer Science*, 1995.

[2] T. Pradhan, A. Israni, and M. Sharma, "Solving the 0-1 Knapsack problem using Genetic Algorithm and Rough Set Theory", *IEEE Inter. Conf. on Adv. Commu., Control and Compu. Tech.*, pp. 1120-1125, 2014.

[3] C. Sachdeva and S. Goel, “An improved approach for solving 0/1 Knapsack Problem in polynomial time using Genetic Algorithms”, *IEEE Inter. Conf. on Recent Adv. and Inno. in Engg.*, 2014.

[4] K. Gupta, “A hybrid GA-GSA algorithm to solve multidimensional knapsack problem”, *IEEE Inter. Conf. on Recent Adv. in Info. Tech., 2018.*

[5] A. E. Ezugwu et al., “A comparative study of Meta-Heuristic Optimization Algorithms for 0-1 Knapsack Problem”, *IEEE Access*, pp. 43979-44001, 2019.

[6] O. Marek, “Introduction to Genetic Algorithms”, 1998. https://www.obitko.com/tutorials/genetic-algorithms/

[7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, “Introduction to Algorithms”, *The MIT Press*, 1990.