An Approximation to Multi Dimensional Knapsack problem using Genetic Algorithm

Pradyumna YM
PES1201700986
CSE dept., PES University

Anush V Kini
PES1201701646
CSE dept., PES University

Satvik N Jois
PES1201700213
CSE dept., PES University

***Abstract*—A genetic algorithm is applied to an NP-complete problem, the 0/1 multiple knapsack problem. The partitioning of the search space resulting from this highly constrained problem may include substantially large infeasible regions. Our implementation allows for the breeding and participation of infeasible strings in the population. We apply our genetic algorithm to problem instances from the literature of well-known test problems and report our experimental results. These encouraging results, especially for relatively large test problems, indicate that genetic algorithms can be successfully used as heuristics for finding good solutions for highly constrained NP-complete problems.Genetic algorithm helps us find the optimal solution for 0/1 Multidimensional knapsack.**

I. Introduction

A. The 0/1 Multidimensional Knapsack Problem

The Multidimensional Knapsack Problem is a generalisation of the standard 0-1 Knapsack Problem where instead of considering only one knapsack, one tries to fill m knapsacks of different capacities.

Let $N = 1, .., n$ be the set of items where each item j has a corresponding profit p_j and weight w_j . We consider m knapsacks of capacity c_i, $i \in 1, ., m$, then Multi dimensional knapsack consists in filling all knapsacks so that the total profit is maximised and the sum of weights in each knapsack i does not exceed the capacity c_i. xij ∈ 0, 1, i 1, . ,m, j 1,., n; where pj , ci and wj are positive integers and ensure that the filling of knapsack i does not exceed its corresponding capacity ci and every selected item is assigned only to one knapsack, respectively. The MKP is formulated as the following -1 integer programming problem:

We also note that it can be thought as a resource allocation problem, where we have m resources and n objects. Each resource has its own budget (knapsack capacity), and w ij represents the profit, while working within a certain budget.

B. Genetic Algorithm

The genetic algorithm is a random-based classical evolutionary algorithm. By random here we mean that in order to find a solution using the Genetic algorithm, random changes applied to the current solutions to generate new ones. Genetic algorithm is based on Darwin's theory of evolution. It is a slow gradual process that works by making changes to the making slight and slow changes. Genetic algorithm makes slight changes to its solutions slowly until getting the best solution.

$$\text{maximise : } \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij},$$

$$\text{s.t.} \quad \sum_{j=1}^{n} w_j x_{ij} \le c_i, \; i \in \{1, \dots, m\},$$

$$\sum_{i=1}^{m} x_{ij} \le 1, \quad j \in \{1, \dots, n\},$$

Genetic algorithm works on a population consisting of some solutions where the population size is the number of solutions. Each solution is called individual. Each individual solution has a chromosome. The chromosome is represented as a set of parameters that defines the individual. Each chromosome has a set of genes. Each gene is represented by somehow such as being represented as a string of 0's and 1's. Each individual has a fitness value. To select the best individuals, a fitness function is used. The result of the fitness function is the fitness value representing the quality of the solution. The higher the fitness value the higher the quality the solution. Selection of the best individuals based on their quality is applied to generate what is called a mating pool where the higher quality individual has higher probability of being selected in the mating pool. The individuals in the mating pool are called parents. Every two parents selected from the mating pool will generate two offspring. This will eliminate the bad individuals from generating more bad individuals. By continuing choosing and mating high-quality individuals, there will be higher chances to just keep good properties of the individuals and leave out bad ones. Finally, this will end up with the optimal or acceptable solution.

But, the offspring at present produced utilizing the selected parents simply have the attributes of its parents without any changes. There is no new added to it and in this way, similar downsides in its parents will really exist in the new offspring. To conquer such an issue, a few changes will be applied to every offspring to make new people. The arrangement of all recently produced people will be the new populace that replaces the recently utilized old populace. Every populace made is known as a generation. The process of replacing the old population by the new one is called replacement.

II. Implementation

We have n objects, m constraints, and a limit on the sum of each constraint of each item in the knapsack. Each chromosome in our population is a bit string of length n, where n is the number of objects available. 0 in the i^{th} position means that the item has not been included in the knapsack. Various parameters of the GA have been explained below:

- **Selection:** Candidates are selected for crossover using roulette wheel method. The candidates' fitness values are calculated in terms of percentage, and sectors are allocated to candidates based on their fitness percentages. Next, a random number between 0 and 360 is selected, and the candidate to which the sector belongs is selected for crossover.
- **Crossover:** One point crossover was chosen in order to produce offspring candidates for the next generation. A crossover probability of 0.6 was chosen.
- **Mutation:** a bit flip mutation was used to mutate candidates. A bit flip probability of 0.1 was used, and the probability of a mutation of a candidate was chosen to be $1/(numberofobjects)$.
- **Candidate generation:** for the initial population, the candidates were generated using a random number generator with a slightly higher bias for 0. This was done to prevent the entire population from being infeasible, which would cause the genetic algorithm to fail.
- **Fitness Function:** The fitness function that was chosen to maximise was the sum of the values of all the objects in the knapsack, with a penalty term for candidates which have weight $>$ capacity.
- **Population Size:** A population size of 50 was used.

III. Experiments

An initial population size of 50 was used. A grid search was performed on various hyper parameters. The results of the same have been tabulated below.

A. Iterations

As we see from Fig. 1, the error rate decreases with the number of iterations. So, we used 150 iterations of the genetic algorithm.

B. Population

The genetic algorithm consists of a set of individuals called the population. Each individual represents a solution to the problem to be solved. The number of individuals can be varied to get better results. Hence, we measured the error rate for different population values and visualized the same. The visualization can be seen in Fig 2.

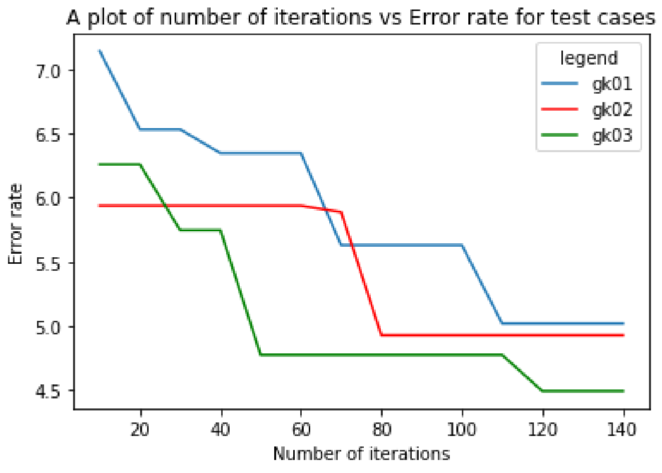

Fig. 1. Iterations vs Error Rate

C. Mutation Rate

The error rates reduce significantly with an increase in the mutation rate upto $p = 0.4$. A mutation rate within the candidate was chosen to be 0.4

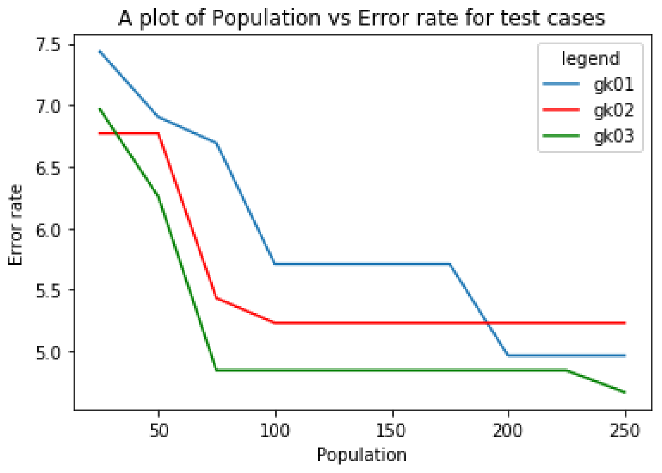

Fig. 2. Population size vs error rate

IV. Results

We have tested the algorithm against different benchmarks online.

TABLE I
Optimal values of the benchmarks

gk01	gk02	gk03
3766	3958	5656

The results we obtained on the benchmark instances have been tabulated below:

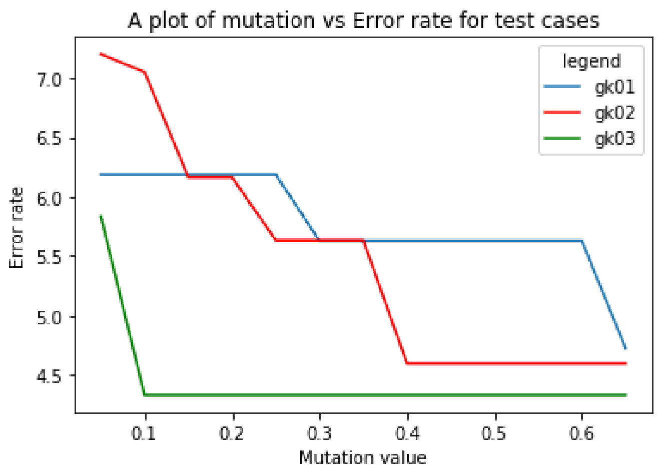

Fig. 3. Mutation rate vs Error Rate

TABLE II
A TABULATION OF TOP 5 RESULTS OBTAINED FOR DIFFERENT BENCHMARKS

gk01	**gk02**	**gk03**
3588	3769	5401
3576	3761	5391
3575	3758	5387
3572	3749	5384
3568	3745	5383

V. CONCLUSION

The multidimensional knapsack problem is a NP complete problem that takes combinatorial amount of time to find the optimal solution. Using genetic algorithm, we can estimate the solution reasonably well using a genetic algorithm.