Solving 0-1 Knapsack Problem Using Genetic Algorithms

Rattan Preet Singh[1]
[1]University School of Information Technology (GGS Indraprastha University)
New Delhi, India
E-mail: rattan1208@gmail.com

Abstract -**This is a research project on using Genetic Algorithm to solve 0-1 Knapsack Problem. Knapsack problem is a combinational optimization problem. Given a set of items, each with a weight & value, it determine the number of each item to include in a collection so that the total weight is less than a given limit & the total value is as large as possible. The paper consists of three parts. In the first section we give brief description of Genetic Algorithms and some of its basic elements. Next, we describe the Knapsack Problem and Implementation of Knapsack problem using Genetic Algorithm.**

The main purpose of this paper is to implement Knapsack problem by an algorithm that is based on Genetic Algorithm. In this paper I have used Roulette-Wheel, Tournament Selection and Stochastic selection as a selection function and the succeeding populations are analyzed for the fitness value with hope to achieve the correct solution and expected results were observed.

Keywords-Genetic Algorithms, Knapsack Problem, Selection operators

I.INTRODUCTION

In this paper Genetic Algorithm is used to solve the 0-1 Knapsack problem which aims to maximize the weights of objects to be placed in a Knapsack of fixed capacity. Knapsack is a class of NP problem which cannot be solved in linear amount of time however its solution can be verified in linear time. Many approaches have been there to solve this problem namely Dynamic Programming, Greedy Method etc but they are not much efficient. The complexity of Dynamic approach is of the order of $O(n^{3})$ whereas the Greedy Method doesn't always converge to an optimum solution.

So Genetic Algorithm may prove to have an edge over these traditional methods and may lead to solution in more efficient way.

II.GENETIC ALGORITHMS (GA'S)

Genetic Algorithm mimics the process of natural evolution to find a solution of a problem from a set of solutions called population. It belongs to a class of Evolutionary Algorithms and is generally used to find solutions for search and optimization problems. The concepts that are applied in GA's that are inspired by natural evolution are:

1. Inheritance
2. Mutation
3. Selection
4. Crossover

Genetic algorithm begins with a population of strings (called chromosomes or the genotype of the genome), which encode candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem. Then new populations are created from the old population with more representative of solution having more fitness value thus new population evolves toward better solutions. Solutions are generally represented in binary form as strings of 0s and 1s, but other encodings can also be used. The algorithm generally begins from a population of randomly generated individual. In each generation, the fitness of every individual in the population is evaluated and multiple individuals are selected from the current population (based on their fitness), and modified (recombined and possibly randomly mutated) to form a new population. The new population is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached.

A typical genetic algorithm requires:

1. A genetic representation of the solution domain,
2. A fitness function to evaluate the solution domain
3. A reproduction method favoring solutions with more fitness value.

Some of the basic elements of Genetic Algorithms are:

Chromosomes: A chromosome basically consists of a set of parameters that defines a solution to a proposed problem. A chromosome is generally represented in form of a string may be in binary or any other notation but other data structures can also be used.

Fitness function: Fitness function is objective used to evaluate the optimality of a solution. Fitness function predicts the probability of a chromosome to be passed in the next generation. It ranks a chromosome in relation to other chromosomes on basis of its fitness value thus defining its chances of survival and to be passed to next generation. We mention here probability because it is not mandatory that all the chromosomes are passed to next population.

Basic steps of Genetic Algorithm are:

978-1-61284-486-2/11/$26.00 ©2011 IEEE

1: Initialisation

2: Selection

3: Reproduction

4: Termination

Initialisation-In this step genetic algorithm begins by randomly generating a set of instances called chromosomes defining the initial population. Size of initial population varies depending upon the nature of the population.

Selection- In this step members of present populations are selected based on their fitness value to reproduce another generation of population. The chromosomes having higher fitness value is more likely to survive and may contribute several times to the new population. The chromosomes with low fitness value will have fewer members in the new population or some of them may get eliminated also. The size of population remains fixed in each population. The selection process is random in nature. Some of the popular selection methods are:

1: Roulette-wheel

2: Tournament selection

3: Stochastic selection

4: Remainder Selection

Reproduction - In this step the next generation of population is created using the following genetic operators:

1: Crossover - In crossover new offspring's are produced from a pair of chromosome called parent. In this process first a specific position 'p' along the string of chromosome is selected such that 1<=p<l, where l is the length of the chromosome string. After this all the bit positions for p+1 to l are swapped b\w the pair of chromosomes to produce two new child chromosomes. In this way inherit traits of both their parents and have more chances of survival. For e.g.:

Crossover site

Parent1: 0 0 1 0| 0 0 1 0 1

Parent 2: 1 1 0 1| 1 0 1 0 0

Child 1: 0 0 1 0 1 0 1 0 0

Child 2: 1 1 0 1 0 0 1 0 1

In above example each child inherits first four bits from one parent and the other four from the other one and thus having characteristic of each of its parents.

2: Mutation - It simply means changing a bit from 0 to 1 or 1 to 0. Mutation is done to prevent some random loss of potentially useful solution due to reproduction or crossover. It is done with a very small probability of about one mutation per thousand bit of transfer. For e.g.:

Chromosome: 1 0 0 1 1 0 1 0

Mutated Chromosome: 1 0 0 1 1 1 1 0

In above example mutation occurred at bit 6 of the chromosome.

Termination - The process of selection and reproduction are repeated continuously until some termination is reached. These conditions may be one of the following:

- A solution is found that satisfies minimum criteria
- Maximum number of generations reached
- Time allocated has expired
- Combinations of the above

Simple Genetic Algorithm Pseudo Code

1. Choose the initial population of individuals
2. Evaluate the fitness of each individual in that population
3. Create a new population by the following steps :
   - Selection
   - Reproduction (Mutation , Crossover)
4. Replace the present population with the new one.
5. If termination condition is satisfied then exit otherwise go to step 2

III.KNAPSACK PROBLEM

The Knapsack Problem is a combinatorial optimization Problem. Given a set of items with a weight and value, it seeks to select a number of items to be placed in a Knapsack of fixed capacity such that total weight of the items are less or equal to the capacity but its value is as large as possible. The problem often arises in resource allocation with financial constraints.

Let there are N item available and V_i , Q_i and W_i represents the value, quantity and weight of i^{th} item respectively. Let C be the capacity of Knapsack.

The aim of the Knapsack problem is to:

$$\sum_{i=1}^{n} \max D_i V_i$$

where D_i is quantity of item i to be included in knapsack.

Such that:

$$\sum_{i=1}^{n} D_i W_i \leq C$$

$$0 \leq D_i \leq Q_i$$

Example of a 0-1 Knapsack Problem

Consider a Knapsack having a capacity of 10 and following four items are available.

Item	Value	Weight
A	42	7
B	12	3

C 40 4
D 25 5

The aim of the Knapsack problem is to:

$$\sum_{i=1}^{4} \max D_i V_i = 42D_1 + 12D_2 + 40D_3 + 25D_4$$

Where $D_i \in \{0, 1\}$ and

$$\sum_{i=1}^{4} D_i W_i = 7D_1 + 7D_2 + 4D_3 + 5D_4 \leq 10$$

There are 2^4 solutions possible for the above problem:

TABLE 1. SAMPLE SOLUTIONS

A	B	C	D	Weight of the set	Value of the set
0	0	0	0	0	0
0	0	0	1	5	25
0	0	1	0	4	40
0	0	1	1	9	65
0	1	0	0	3	12
0	1	0	1	8	37
0	1	1	0	7	52
0	1	1	1	Weight exceeded	-
1	0	0	0	7	42
1	0	0	1	Weight exceeded	-
1	0	1	0	Weight exceeded	-
1	0	1	1	Weight exceeded	-
1	1	0	0	10	54
1	1	0	1	Weight exceeded	-
1	1	1	0	Weight exceeded	-
1	1	1	1	Weight exceeded	-

Hence for the above a problem 16 number of solutions are possible but there is only one optimum solution having value=65 when A=0 and B=0 and C = D= 1. This means the value of items are maximized when object C and D is placed in the knapsack.

IV.IMPLEMENTATION OF 0-1 KNAPSACK PROBLEM USING GA

Representation of items - Items can be represented using a 2D Array having two columns containing value and weight of items respectively.

I [ ][ ]=

	1	2
1	42	7
2	12	3
3	40	4
4	25	5
	Values	Weight

Representations of Chromosomes -The number of bits in the chromosomes will be equal to the number of items with i^{th} bit denoting the presence of the i^{th} item in the Knapsack. If i^{th} bit is 0 means the item is not present in the knapsack and value 1 indicates its presence. For e.g. consider the following chromosome:

Chromosome: 1 0 1 0

The above chromosome corresponds to the presence of 1^{st} and 3^{rd} item in the Knapsack.

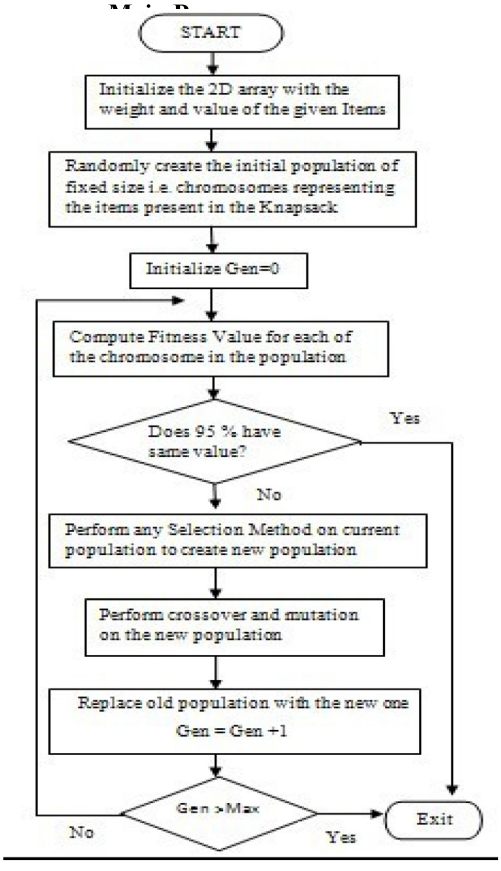


Figure 1. Algorithm for simulation

Fitness Function - Let F (D) defines a fitness function:

$F(D) = I_{[1][1]} D_{[1]} + I_{[2][1]} D_{[2]} + I_{[3][1]} D_{[3]} + \ldots\ldots\ldots + I_{[n-1][1]} D_{[n-1]} + I_{[n][1]} D_{[n]}$

Constraint - Let G (D) defines a Constraint function:

$G(D) = I_{[1][2]} D_{[1]} + I_{[2][2]} D_{[2]} + I_{[3][2]} D_{[3]} \ldots\ldots + I_{[n-1][2]} D_{[n-1]} + I_{[n][2]} D_{[n]} \leq C$

I = 2D Array containing the Weight of items in the second column
D= Chromosome bit string where D_n refers to the bit value of n^{th} bit.
C= Capacity of the Knapsack
Termination Condition - The program terminates when the 95% of the chromosomes have same fitness value or when the number of generations exceeds the predefined upper limit.

Now let us apply Genetic Algorithm on the following Knapsack Problem

Consider the previous Knapsack Problem having a Knapsack capacity of 10 and four items:

2D Array containing Value and Weight of each Item

I [ ][ ]=	1	2
1	42	7
2	12	3
3	40	4
4	25	5
	Values	Weight

Fitness Function

$F(D) = 42 D_{[1]} + 12 D_{[2]} + 40 D_{[3]} + 25 D_{[n]}$

Constraint

$G(D) = 7 D_{[1]} + 3 D_{[2]} + 4 D_{[3]} + 5 D_{[n]} \leq 10$

As we Know the solution of above Problem is: 0 0 1 1

This means the value of items are maximized when item 3 and 4 is placed in the Knapsack.

On solving above problem in Mat lab using different Selection Methods of Genetic Algorithm we obtained the following results:

Stochastic selection

Generation	f(x)	constraint
1	-65	1
2	-107	1
3	-65	1
4	-65	1
5	-65	1

Optimization terminated

Roulette-wheel Selection

Generation	f(x)	constraint
1	- 65	1
2	-65	1
3	-65	1

Optimization terminated.

Tournament Selection

Generation	f(x)	constraint
1	-65	1
2	-107	1
3	-65	1
4	-65	1
5	-65	1

Optimization terminated

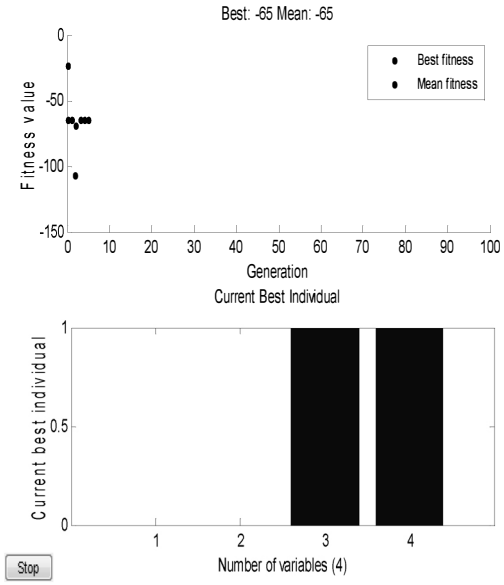


Figure 2. Output using stochastic selection

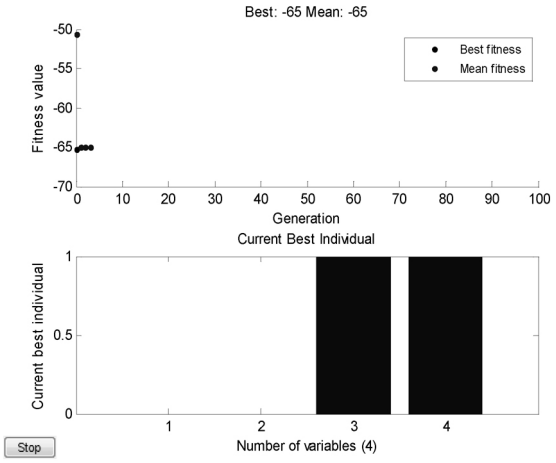
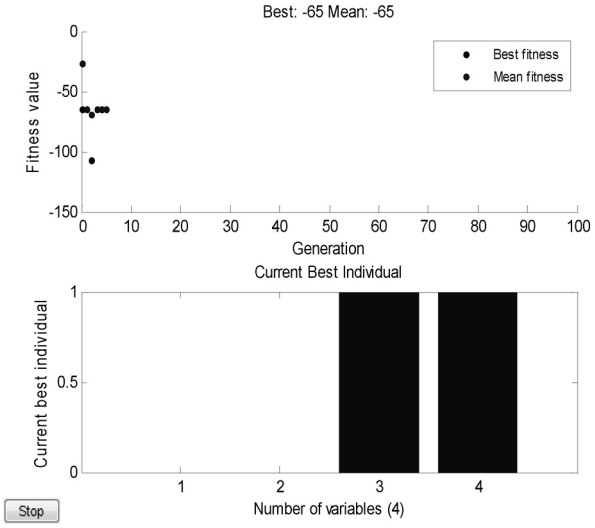


Figure 3 Output using roulette-wheel selection



.Figure 4. Output using tournament selection

V.CONCLUSION

Genetic Algorithms can be used to solve the NP Complete Knapsack Problem. Traditional methods like "Dynamic Programming" which is used to solve Knapsack Problem have exponential complexity. But Genetic Algorithms provide a way to solve the above problem in linear amount of time.

ACKNOWLEDGMENT

I would like to acknowledge the assistance of Assistant Professor Mrs. Jyotsna Yadav in the preparation of this paper.

REFERENCES

[1] Genetic Algorithms in Search, Optimization and Machine Learning – David E. Goldberg.

[2] An Introduction To Genetic Algorithm- Melanie Mitchell.