Parallel Dual Population Genetic Algorithm for Solving 0/1 Knapsack Problem

Parallel Computing(IT300) Project Report

Submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

In

INFORMATION TECHNOLOGY

by

Akshay Khatri (171IT206)

Shashank Jaiswal (171IT239)

DEPARTMENT OF INFORMATION TECHNOLOGY

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA

SURATHKAL, MANGALORE - 575025

SEPTEMBER, 2019

CERTIFICATE

This is to certify that the Parallel Computing (IT300) Project entitled **"Parallel Dual Population Genetic Algorithm for Solving 0/1 Knapsack Problem"** has been presented by Akshay Khatri and Shashank Jaiswal, students of V semester B.Tech. (I.T.), Department of Information Technology, National Institute of Technology Karnataka, Surathkal, on the 25th of September, 2019, during the odd semester of the academic year 2019 - 2020, in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Information Technology.

Mr. Rajat

Signature of Examiner-1 with Date

Dr. Neelima Bayyapu

Signature of Guide with Date

Place: NITK, SURATHKAL

Date: 25th September, 2019

DECLARATION

We, Akshay Khatri and Shashank Jaiswal, hereby declare that the project entitled “Parallel Dual Population Genetic Algorithm for Solving 0/1 Knapsack Problem” was carried by us during the V semester of B.Tech(I.T.) , academic year 2019 - 2020. We declare that this is our original work and has been completed successfully according to the direction of our guide Dr. Neelima Bayyapu and as per the specifications of NITK Surathkal.

Place: NITK, SURATHKAL

Date: 25th September, 2019 Signature of Students

ABSTRACT

Dual Population Genetic Algorithm is an effective optimization algorithm that provides additional diversity to the main population. It deals with the premature convergence problem as well as the diversity problem associated with Genetic Algorithm. But dual population introduces additional search space that increases time required to find an optimal solution. This large scale search space problem can be easily solved efficiently using all available cores of current age multi-core processors. Results of Sequential DPGA and Parallel DPGA are compared for the 0/1 Knapsack problem on the basis of accuracy and run time.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1 INTRODUCTION

In this paper, Dual Population Genetic Algorithm with parallel environment is used to solve 0/1 Knapsack problem which aims to maximize the weights of objects to be placed in a Knapsack of fixed capacity to get maximum profit. Knapsack problem is an NP-complete problem which cannot be solved in polynomial 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 Programming approach is of the order of O(nW) where n is the number of objects and W is the weight of sack, whereas the Greedy Method doesn't always converge to an optimum solution.

One way to get solution of above problem is to use Genetic Algorithm[1]. But genetic algorithm may converge to a local optimum solution. So, to avoid convergence to local optima, we will use Dual Population Genetic algorithm. Dual-population GA (DPGA) can be viewed as a type of multipopulation GA in which it manipulates two isolated populations. The additional population only plays the role of a repository for maintaining diversity or additional information.

Dual Population Genetic Algorithm[3] requires two sets of population which increases search space and memory space. To solve this problem, we will python's library **concurrent.futures**. The concurrent.futures modules provides interfaces for running tasks using pool of thread or process workers. The APIs are the same, so applications can switch between threads and processes with minimal changes. The module provides two types of classes for interacting with the pools. **Executors** are used for managing pools of workers, and **futures** are used for managing results computed by the workers.

Chapter 2 LITERATURE REVIEW

2.1 Background

Dual Population Genetic Algorithm is an open research problem. Park and Ruy (2006) introduced DPGA. It has two distinct populations with different evolutionary objectives: The Prospect (main) population works like population of the regular genetic algorithm which aims to optimize the objective function. The Preserver (reserve) population serves to maintain the diversity. Umbarkar and Joshi (2013) compared DPGA with OpenMP GA for Multimodal Function Optimization. The results show that the performance of OpenMP GA better than SGA on the basis of execution time and speed up.

2.2 Problem Statement

Dual Population Genetic Algorithm is an effective optimization algorithm that provides additional diversity to the main population. It deals with the premature convergence problem as well as the diversity problem associated with Genetic Algorithm. But dual population introduces additional search space that increases time required to find an optimal solution. The problem is to design a solution for the 0/1 Knapsack problem such that it utilizes all the available cores of the modern-day CPUs and speeds up the computation.

2.3 Objectives

- Program a solution for the 0/1 Knapsack problem using Dual Population Genetic Algorithm which yields better convergence compared to Genetic Algorithm.
- Exploit the advantages provided by parallel environment to use all the cores of the processor and gain a speed up compared to the sequential algorithm.

Chapter 3 LACUNA IN THE EXISTING PROBLEM

0/1 Knapsack problem is a constraint optimisation problem. Solving 0/1 Knapsack problem with genetic algorithm doesn't always give best possible solution. It may converge to a local optimum solution due to lack of diversity available in the population set. So , to get better convergence we are using DPGA. Problem with using DPGA is that, it increases search space as there are now two sets of population. Because of the increased search space, more computations are required. Parallel programming environment can be used to parallelize the above problem so as the increase in search space doesn't severely affect the time consumption to solve the problem.

Chapter 4 MOTIVATION

0/1 Knapsack Problem is an NP-complete problem for which solution doesn't exists in polynomial time. Using Genetic Algorithm[4], the optimal solution for the same can be obtained in polynomial time. DPG algorithm can be used for better convergence. But it will increase the search space , and thus more computations are required which will increase both time and memory space. This hurdle can be eliminated using various parallel environment constructs, which will decrease computation time and provide quicker convergence and more effective utilisation of resources.

Chapter 5 METHODOLOGY

5.1 Genetic Algorithm:

The genetic algorithm is a method for solving both constrained and unconstrained optimization problems that is based on natural selection, the process that drives biological evolution. The genetic algorithm repeatedly modifies a population of individual solutions. At each step, the genetic algorithm selects individuals at random from the current population to be parents and uses them to produce the children for the next generation. Over successive generations, the population "evolves" toward an optimal solution. You can apply the genetic algorithm to solve a variety of optimization problems that are not well suited for standard optimization algorithms, including problems in which the objective function is discontinuous, nondifferentiable, stochastic, or highly nonlinear. The genetic algorithm can address problems of *mixed integer programming*, where some components are restricted to be integer-valued.

5.2 Dual Population Genetic Algorithm (DPGA):

A dual-population genetic algorithm (DPGA) is a new multi-population genetic algorithm that solves problems using two populations with different evolutionary objectives. The main population is similar to that of an ordinary genetic algorithm, and it evolves in order to obtain suitable solutions. The reserve population evolves to maintain and offer diversity to the main population. The two populations exchange genetic materials using interpopulation crossbreeding.

5.3 concurrent.futures (Launching parallel tasks):

The concurrent.futures module in python provides a high-level interface for asynchronously executing callables. The asynchronous execution can be performed with threads, using ThreadPoolExecutor, or separate processes, using ProcessPoolExecutor. Both implement the same interface, which is defined by the abstract Executor class.

The concurrent.futures.Executor is an abstract class that provides methods to execute calls asynchronously. It should not be used directly, but through its concrete subclasses. Its submit method schedules the callable passed as an argument, to be executed and returns a Future object representing the execution of the callable. The results returned by executing the callable can be obtained by using the method result of the Future object.

Chapter 6 IMPLEMENTATION STRATEGY

- First we will write sequential 0/1 Knapsack problem using Genetic Algorithm.
- Then we will identify parallelism in above algorithm.
- We will use python parallel programming constructs for task and data sharing.
- As Fitness Evaluation of each individual is an independent step , it can be executed in parallel. The python library concurrent.futures parallel directives will be used to create multiple threads. These threads will evaluate each individual's fitness separately.
- Better convergence to the optimal solution of the problem is achieved using DPG algorithm.

6.1 Serial DPGA 0/1 Knapsack Pseudo Code

- begin
- Initialize main population M0, reserve population R0, crossover rate, elitism rate, mutation rate, crossbreeding rate, tour size, max generation tmax
- Initialize M0 of size m
- Initialize R0 of size n, n>m
- t: = 0
- Repeat
- Step I: Encoding of both population from decimal value representation to binary value representation
- Step II: Fitness Calculation
  - Evaluate M0 using objective function fm(x)
  - Evaluate R0 using fitness function for reserve population, fr(x)
- Step III: Inbreeding of main population and generate intermediate main population Om via crossover

- Step IV: Inbreeding of reserve population and generate intermediate reserve population Or via crossover
- Step V: Crossbreeding
  - Offspring C of size (n-m) using best individuals from Om and Or
  - Make Im = C U Om and Ir = C U Or where U is Union
- Step VI: Decoding: Decoding of both population from binary representation to decimal representation
- Step VII: Evaluation
  - Evaluate Im using fm(x)
  - Evaluate Ir using fr(x)
- Step VIII: Survival selection from Im of size m and from Ir of size n
- t = t + 1

Until

fm(x) > = global optimal value or t > tmax

End

Where,

t : index of current generation

M0, R0: main, reserve population

fm(x): objective function

Om, Or: intermediate main, reserve population respectively

fr(x): fitness function for reserve population

Im, Ir: constitute set of main, reserve population respectively

C: offspring

tmax: maximum generations

Chapter 7 EXPERIMENTAL SETUP

The experiment was run in phases, each phase corresponding to an increase in problem size. Different test problems of various sizes were randomly generated. Sizes vary from 50(smallest) to 400(largest). Weight and Value array was obtained by using random functions. Population size was also randomly chosen and different sets of populations was obtained. Convergence condition was decided by function named test which checks for the percentage of occurence of mode value in fitness array. If above convergence technique failed , then convergence was decided by running the algorithm for maximum generation times which was again chosen randomly.

Chapter 8 RESULTS AND OBSERVATIONS

- The genetic algorithm runs faster than the trivial Dynamic Programming approach to the 0/1 Knapsack problem for large input sizes.
- Dual Population Genetic Algorithm leads to better convergence to the optimal solution and hence performs better as we have two populations compared to the genetic algorithm. Fig.1 displays the output of the regular Genetic algorithm for an input size of 400 when the maximum weight of the knapsack is 5000. The same for the Dual Population Genetic Algorithm is shown in Fig.2. It is clearly visible that the latter gives better output(better converges to the optimal solution) when compared with the former.

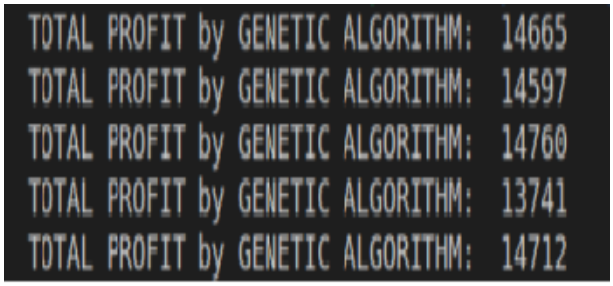


Fig.1: GA Output

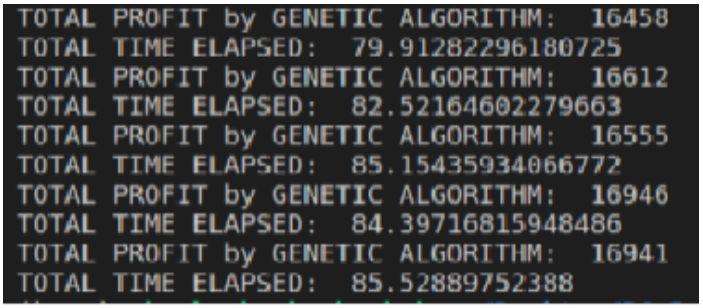


Fig.2: DPGA Output

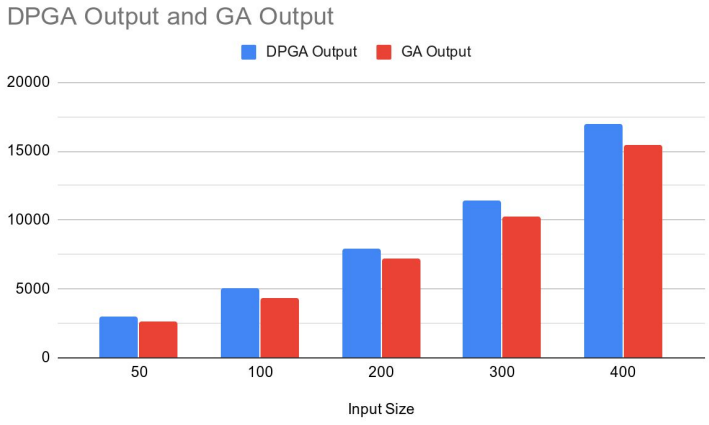


Fig.3: GA output vs DPGA output

- Parallel DGPA performs better than the DGPA algorithm as it uses all the cores for the computation.Fig.4 displays the output of the serial DPGA algorithm for an input size of 200 when the maximum weight of the knapsack is 2000. The same for the parallel version is shown in Fig.5. It is observed that the parallel algorithm takes less time compared to the serial one.

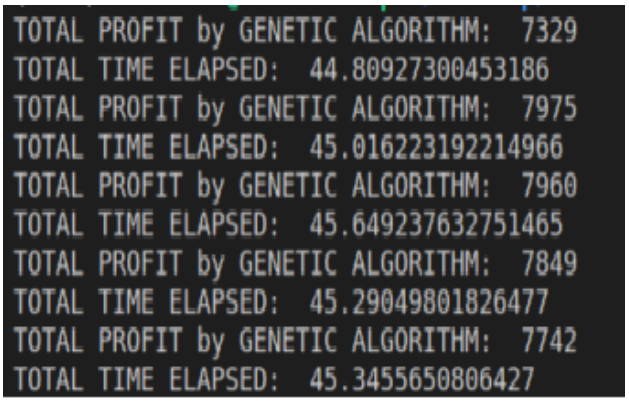


Fig.4: DPGA Execution Time

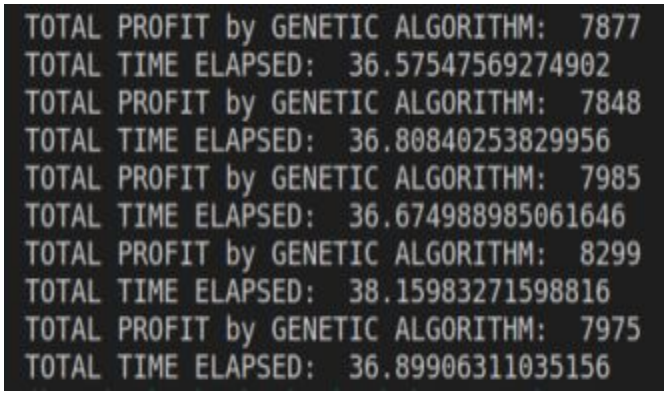


Fig.5: Parallel DPGA Execution Time

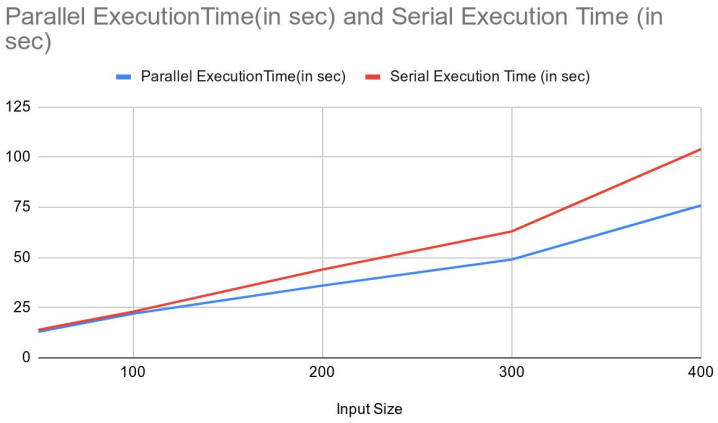


Fig.6 Parallel vs Serial Execution Time(in sec)

Chapter 9 Challenges Involved and Challenges Addressed

Challenges involved:

- Understanding the pattern of using Genetic Algorithm to solve the 0/1 Knapsack Problem.
- Extending the above approach to use Dual Population Genetic Algorithm.
- Finding appropriate regions in the code where parallelisation can be done.
- Finally, using Python parallel programming techniques to implement the same.

Challenges addressed:

- Coming up with an algorithm for cross-breeding between the main population and the reserve population.
- Understanding how the concurrent.futures library in Python helps in parallel programming.
- Finding regions in code where parallelisation is possible and using appropriate constructs for the same.

Chapter 10 CONCLUSIONS AND FUTURE TRENDS

Parallel DPGA(Dual Population Genetic Algorithm) is a novel technique for solving Constrained Optimized Problems(COPs). Its aim is to use all available cores of current age multicore processors. Experiments conducted using CEC 2006 problems set show increase in speed up with increase in no. of processors till certain no. of processors and then speed up remains constant. This behavior emulates Amdahl's Law. The main objective of the project was to design a program that solves the 0/1 Knapsack problem given any weights and values of objects and the maximum weight of the sack that gives a better (more optimum) solution as well as it executes in less time when compared to the sequential Dual Population Genetic Algorithm for solving the same. The parallel DPGA algorithm fulfills the requirement of the problem statement.

In future, using alternative constraints handling method, parallel algorithm and high performance computing paradigm, a better speed up can be achieved. CUDA can be used for obtaining higher speedup as GPUs provide better parallel environment than CPUs.

REFERENCES

[1] Solving 0–1 Knapsack problem using Genetic Algorithms (2011) - https://ieeexplore.ieee.org/document/6013975

[2] A Dual-Population Genetic Algorithm for Adaptive Diversity Control (2010) - https://ieeexplore.ieee.org/document/5491154

[3] Dual Population Genetic Algorithms (2015) - https://www.researchgate.net/publication/307712784_Dual_Population_Genetic_Algorithm_for_Solving_Constrained_Optimization_Problems

[4] Genetic Algorithms - https://www.geeksforgeeks.org/genetic-algorithms/

[5] concurrent.futures documentation https://docs.python.org/3/library/concurrent.futures.html

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:

$$\sum_{max} D_i V_i \quad (i=1 \text{ to } n)$$

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

Such that:

$$\sum D_i W_i \leq C \quad (i=1 \text{ to } n)$$

$$0 \leq D_i \leq Q_i$$

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.

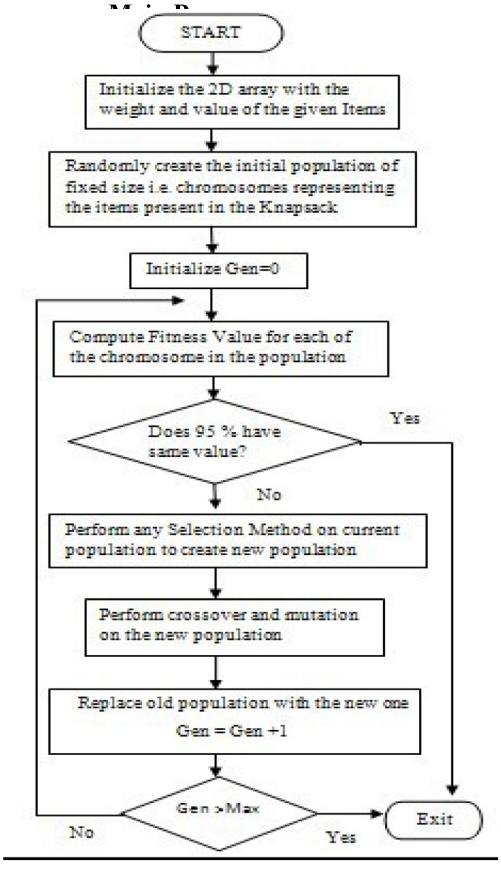


Figure 1. Algorithm for simulation

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

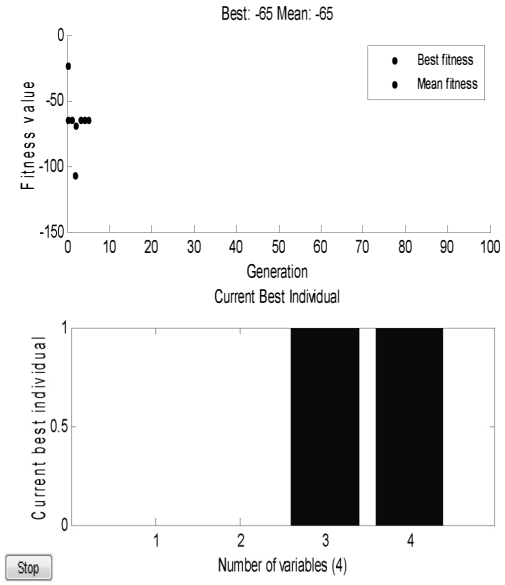


Figure 2. Output using stochastic selection

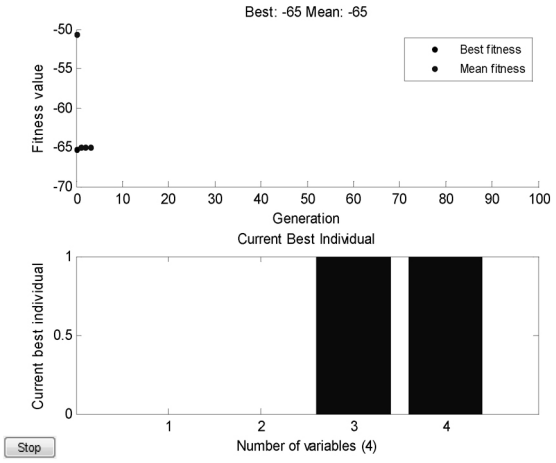
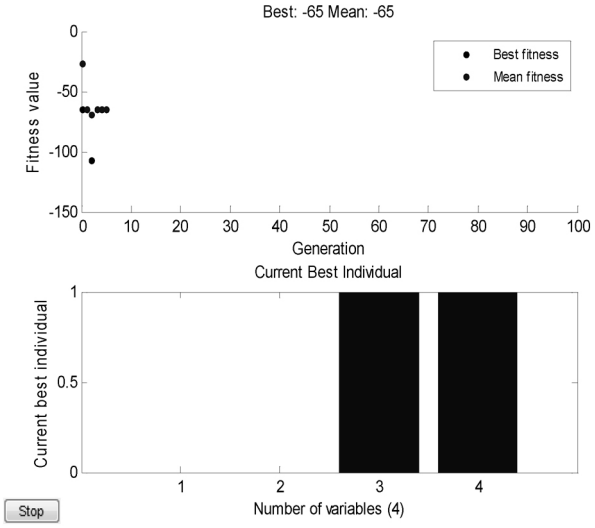


Figure 3 Output using roulette-wheel selection



.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.