An Evolutionary Hybrid Approach for Solving 0/1 Knapsack Problem Optimally in Polynomial Time using Genetic Algorithms

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Computer Science and Engineering

Submitted By
Charu Sachdeva
(Roll No. 801232006)

Under the supervision of:
Dr. Shivani Goel
Assistant Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

June 2014

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, **"An Evolutionary Hybrid Approach for Solving 0/1 Knapsack Problem Optimally in Polynomial Time using Genetic Algorithms"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in Computer Science and Engineering submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of **Dr. Shivani Goel** and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

(Charu Sachdeva)
Roll No. 801232006

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Dr. Shivani Goel)
Assistant Professor
Computer Science and Engineering Department
Thapar University
Patiala

Countersigned by

(Dr. Deepak Garg)
Head
Computer Science and Engineering Department
Thapar University
Patiala

(Dr. S. K. Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

ACKNOWLEDGEMENT

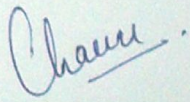
The successful completion of any task would be incomplete without acknowledging the people who made it possible and whose constant guidance and encouragement secured the success.

First of all I wish to acknowledge the benevolence of omnipotent God who gave me strength and courage to overcome all obstacles and showed me the silver lining in the dark clouds.

With the profound sense of gratitude and heartiest regard, I express my sincere feelings of indebtedness to my guide **Dr. Shivani Goel**, Assistant Professor, Computer Science and Engineering Department, Thapar University for her positive attitude, excellent guidance, constant encouragement, keen interest, invaluable co-operation, generous attitude and above all her blessings. She has been a source of inspiration for me.

I am grateful to **Dr. Deepak Garg**, Head of Department, and **Dr. Ashutosh Mishra**, P.G. Coordinator, Computer Science and Engineering Department, Thapar University for the motivation and inspiration for the completion of this thesis.

Last but not the least I would like to express my heartfelt thanks to my parents and my friends who with their thought provoking views, veracity and whole hearted co-operation helped me in doing this thesis.



(Charu Sachdeva)

Roll No. 801232006

ABSTRACT

The 0/1 knapsack is a very well known problem and many approaches have been proposed such as dynamic programming and greedy strategy to solve this problem. But 0/1 knapsack problem is a non polynomial time problem. Solving it in a polynomial time is a challenge. It is becoming an important problem because there are many real life applications based on this. Genetic Algorithms have been proved to be a good approach in solving these types of problem and with the help of Genetic Algorithms it will no longer remain a NP problem. There is a problem in the existing approach for solving 0/1 knapsack problem with the help of Genetic Algorithms and with the existing approach the performance is not good. In this thesis a new improved algorithm is proposed which gives better result than the existing algorithm and improves the performance. A number of numerical experiments are performed and the outcome shows how this approach is better than the previous approach of Genetic Algorithm for solving 0/1 Knapsack Problem.0/1 Knapsack problem now no longer remain NP problem and it can be solved optimally in polynomial time.

TABLE OF CONTENTS

LIST OF FIGURES

List of Tables

CHAPTER 1: INTRODUCTION

1.1 Motivation

"*Intelligence is basically "the capacity to learn and solve problems*"- Webster's dictionary.

Artificial Intelligence is a field in which one can understand and build intelligent entities. It is basically the ability to act rationally and the ability to solve novel problems. Artificial Intelligence has a vast domain. Many subtopics such as Genetic Algorithms, Expert systems, Intelligent Agents, Soft Computing etc. come under it's domain. Artificial Intelligence is involved in reasoning and planning in the way by solving a new problem, planning and making decisions. There are many non polynomial time (NP) problems which are closely related to real world problems. 0/1 Knapsack Problem is one of the NP problem. To solve the above problem in polynomial time and get an optimal solution is itself a challenging task. Existing method can't solve it in polynomial time to get optimal solution but with the help of hybrid approach this problem now no longer remain NP problem and can be solved in polynomial time to get optimal solution.

1.2 Artificial Intelligence

AI is a branch of computer science and is defined as the intelligence of machines or software. AI develops software and intelligent agents. Several technical issues divide AI research. The goals of AI are the ability to manipulate and move objects, learning, reasoning, communication, planning. General Intelligence is known as Strong AI. There are many subfields in AI domain. Soft Computing is one of them. Main characteristics of AI given below [1-3]:

1. **Learning and adaption**

   AI is continuously learning and adapting and also the internal models are always being updated.

2. **Reasoning and Planning**

   AI has the ability to deal with uncertainties and unexpected problems. It continuously looks for solving new problems, planning and making decisions.

3. **Ability to interact with real world**

   AI has an ability to perceive, understand and act. Also have ability for speech reorganization and image processing and an ability to take actions etc.

1.3 Soft Computing

Soft computing uses soft computing techniques for system solution. It supports the combination of soft computing techniques and tools into both advanced as well as everyday applications [4].

The main idea after the soft computing is to model comprehensive behaviour of human mind. It shows conceptual intelligence in machines. It is different from hard computing in the sense that soft computing is permissive of approximation, uncertainty, imprecision, partial truth etc. Hard Computing is also known as conventional computing. There is a lot of difference between soft computing and hard computing as given below [5]:

Table 1.1: Showing difference between Soft Computing and Hard Computing

Hard Computing	**Soft Computing**
It depends upon exactly stared analytical model	It is tolerant of imprecision
It requires full truth	It can work with partial truth
It is accurate and precise	It is Imprecise
It needs high cost for Solution	It needs low cost for solution
Not suitable for real world problems	Suitable for real world problems
Requires a lot of computation time	Requires reasonably less time

1.3.1 Unique Features of Soft Computing

Soft computing can be characterize by its unique features as following [5]:

- In some particular domains it process human like expertise
- It can explain their decisions
- Computationally Intelligent
- It can adapt when there is change in environment and can learn to do better

1.3.2 Components of Soft Computing

Various components of soft computing are:

- Neural Network (NN)
- Machine Learning (ML)
- Fuzzy Logic (FL)
- Evolutionary Computation(EC)
    - Genetic Algorithm
    - Swarm Intelligence
    - Ant Colony Optimization

In present research work, the focus here is on evolutionary computing.

1.3.3 Evolutionary Computation [6]

Evolutionary Computing (EC) is the problem-solving technique which is typically based on biological evolution. Biological Evolution involves natural selection and genetic inheritance. There are varieties of problems on which these techniques can be applied. Problems include various applications of commerce, industrial and scientific research.

1.3.4 Advantages of EC

Many advantages of evolutionary computing are:

- Parallelism
- Solves problems that have no solution
- Broad applicability
- Conceptual simplicity
- Robust to dynamic Changes

1.4 Genetic Algorithms

John Holland was the first introducer of Genetic Algorithms(GA). He introduced genetic algorithms in 1970s as a result of investigations. GAs comes under soft computing paradigm which is known as evolutionary computing. In this the process of solving a problem is very similar to biological evolution. The process involves fitness, selection, crossover, mutation to generate better solutions in form of chromosomes from the existing solutions.

These algorithms involve finding solutions to the problems for which no acceptable algorithmic solution exists. The GA methodology involves optimization. In optimization there is a solution space consisting of large number of solution and one has to search one or more very good solutions for our problem. Genetic Algorithms continuously reduce the search space by eliminating the solutions which ranked as poor, and generate a new generation through crossover and mutation those ranked as good. Ranking is based on the fitness value of each candidate.

There are two ways that differentiate Genetic Algorithm from standard optimization algorithm and can be summarized as [7]:

Table 1.2: Standard Algorithms versus Genetic Algorithms

Standard Algorithms	Genetic Algorithm
A single point is generated in each iteration. Optimal solution is approached by sequence of Points.	A population is generated in each iteration. In Genetic Algorithm population approaches to optimal solution.
Uses Deterministic Computation to select next point in the sequence.	Uses Computations that involve random choices to select next population

1.4.1 Various Terminologies in Genetic Algorithms:

In order to use GAs, terminology used is given below:

- **Search space/ State space:** It is the space of all feasible solutions or all the possible solutions.
- **Gene:** Gene is the smallest unit of genetic algorithm. It represents a unit of information.
- **Chromosome:** Chromosome is a series of genes. Chromosome can be considered one possible solution to the problem. In the solution pattern each gene in the chromosome represents one component.

- **Encoding:** Representing chromosome in a particular form for e.g. 0 or 1, alphabetical etc.
- **Population**: A set of solutions or individuals or chromosomes.
- **Generation**: The process of evaluation, selection, recombination (crossover) and mutation.
- **Fitness:** A fitness function is an objective function. It is value assigned to an individual or chromosome based on how far or close it is from the solution. If fitness value is greater, then it contains a better the solution.
- **Allele:** Value which a gene can assume. It can be binary, integer etc.
- **Selection:** The performance of algorithm significantly is influenced by this operator. Selection is the process in which individual chromosome is selected from population and then crossover and mutation are performed.
- **Crossover:** Crossover is the primary search operator in genetic algorithms. Crossover gives new combinations only by rearranging existing characteristics. It takes two chromosomes as parent and then produces two child chromosomes.
- **Mutation:** Mutation is a random adjustment operator in genetic algorithms. Those characteristics that are not achieved by crossover alone, mutation is useful for introducing that in population.
- **Termination:** Termination, the last step in the genetic algorithms, is the condition according to which genetic algorithm decides whether to continue with the whole procedure again or it should stop the search.

1.4.2 Advantage of Genetic Algorithms

- These are also applicable when little knowledge is encoded in the system
- For the complex problem it is an effective way to find a reasonable solution
- Easy Implementation
- Parallelism
- NP-complete problems are also solved for finding solution in an efficient way

1.4.3 Criteria for GA Approaches

Following criteria can be used for verifying the GA approaches:

- **Completeness:** Completeness here specifies that every solution have its encoding.
- **Characteristics Perseverance:** The useful characteristics from parents should be inherited in offspring or child chromosomes.
- **Soundness:** A corresponding solution should be there for the code produced by genetic operators.

1.5 Non-Deterministic Polynomial Problems

NP (non-deterministic polynomial) problems are such problems for which there are no possible algorithms that would guarantee to run the problem in a polynomial time. However, it is possible to make a guess and check it as a solution. This can be done in polynomial time. Most well known NP problems are Clique, Travelling salesman problem and 0/1 knapsack problem etc. [8].

1.6 0/1 Knapsack Problem

The 0-1 Knapsack problem is a well known problem and it has been studied extensively from the past years. The reason behind this is that there are many real domains in which it appears. Although its NP-completeness, many algorithms have been proposed such as greedy and dynamic that exhibit impressive behavior in the average case. Originally there is only Knapsack Problem, 0/1 Knapsack Problem is a special case of Knapsack problem where 1 indicates item will included in knapsack and 0 indicates it will not be included in knapsack.

The 0/1 knapsack problem is a problem in combinative optimization. In this problem there is a given a set of items each item is associated with a cost and a value. One has to find the items that will be included in the knapsack so that total weight should be less than or equal to weight of the knapsack and total value is as large as possible [9].

Knapsack problem is well known NP problem. It can't be solved in a polynomial amount of time. So there are various approaches for solving 0/1 Knapsack problem such as greedy technique, backtracking, dynamic programming etc.
The methods such as Dynamic Programming considers all the possible solution for solving 0/1 knapsack problem and it gives optimal solution but it can't take polynomial time to solve the problem. Similarly Greedy Strategy takes optimal time

to solve 0/1 knapsack problem but it can't guarantee to give optimal solution for this. So there is a need of a method to solve this in a polynomial time and gives an optimal solution. So one can use Genetic Algorithms to solve 0/1 knapsack problem and can get optimal solution.

1.7 Thesis Outline

The thesis has been organized into 6 chapters including this introductory chapter which briefly outlines Artificial Intelligence, Soft computing, Evolutionary Computing, Genetic Algorithms, NP problems and 0/1 Knapsack Problems.

Chapter 2 presents a literature review and explains the various existing approaches by which 0/1 knapsack problem can be solved i.e. Dynamic Programming and Greedy Strategy. Also explains various operators in genetic algorithms also with some advanced crossover operators and comparison between MHAGA and AGA.

Chapter 3 describes the problem statement including gaps in study and also existing approach to solve 0/1 knapsack problem along with problem found in existing approach. Also describes how proposed algorithm can remove the existing problem.

Chapter 4 describes the proposed algorithm and its implementation with the help of algorithms.

Chapter 5 describes the experimental results to show how the problem can be solved in polynomial time and how the proposed algorithm gives better results with comparing existing algorithm.

Chapter 6 summarizes the conclusion and the scope for future work.

CHAPTER 2: LITRATURE SURVEY

2.1 Non-Deterministic Polynomial Problems

NP problem states that if a problem A is NP and there is another problem B. If polynomial time algorithm for A can also be used to solve other problem B in polynomial time, them the problem B is also a NP problem [10].

Also NP problems are such type of problems which can be solved by applying brute-force to find a solution for the problem, because it will take trillions of years or longer to find the correct solution for the correct or optimal solution for the problem [11].

Currently a standard computer can't solve NP problem in a polynomial amount of time, although simulation of non-determinism is possible but problem is that simulation takes exponential time i.e. with the help of simulation problems can be solved in exponential time. So when the problem size grows it will take large amount of time according to its size. These types of problems can be solved with the help of nondeterministic Turing Machine.

Some of the well known NP problems are 0/1 Knapsack Problem, Sum of Subset Problem, Travelling Salesman Problem, Graph clique Problem etc.

2.2 Polynomial Problems

Polynomial Time Problems are such type of problems which can be solved in polynomial amount of time or these are the problems which can be solved in O (n^k) amount of time, where k is a non-negative integer and n shows the input complexity [12].

Some of the Polynomial type problems are Fibonacci series, Quick Sort, Merge Sort, Linear Search, Binary Search etc. Each of these problems can be solved in a polynomial amount of time.

2.3 0/1 Knapsack Problem

One of the mathematical models for 0-1 programming problem is Knapsack Problem. It is a NP Problem. Knapsack Problem consists of list of N Items. Each item is

associated with weight W_i(i=1,2,3,…….,n) and profit value P_i(i=1,2,3,…….,n) and those items are to be selected so that the following objectives are fulfilled.

- Profit should be maximum
- Total weight should not exceed the capacity of knapsack

Suppose the weight of the knapsack is A pounds i.e. Knapsack can accommodate maximum A pounds.

If the item i included in knapsack then the variable X_i=1, otherwise X_i=0.

The mathematical model for the 0/1 Knapsack as follows:

$$\text{Max } f(X_1, X_2, X_3, \ldots\ldots\ldots\ldots, X_4) = \sum P_i X_i \text{ ,where i=1 to n} \quad (1)$$

$$\sum W_i X_i < A \quad \text{where i=1 to n} \quad (2)$$

$$X_i \in \{0,1\} \quad (3)$$

Equation 1 describes the maximum profit of the items selected. Equation 2 apply the constraint that total weight of the selected items should not exceed the knapsack weight i.e. A pounds. Equation 3 shows that whether the item is selected or not. If 0 then item is not selected and if 1 item is selected in knapsack[13][14]. In this way items are selected and included in the knapsack.

2.3.1 Various applications of 0/1 Knapsack problem[15]:

0/1 knapsack problem can be used in many real life applications like:

- They appear in many industrial domain such as budget allocation, logistics, telecommunication, production management, transportation, cutting and packing, advertisement, reliability, investment etc.
- Knapsack problem appears in the field of optimal research in integer programming. There are many real world problems which are related to knapsack problem in Industry and financial management. For example portfolio management, cutting stock, capital budgeting, production scheduling etc.

- It forms the basis for public key encryption.
- It can be applied in the domain of real-world decision-making processes such as selection of assets for securitization, finding the least wasteful cutting of raw materials etc.
- It can be used in the situations where there is a fixed budget and decision-maker have a sequence of option each associated with cost and value which arise in many contexts. For example bidding in sponsored search auctions, hiring workers, scheduling jobs. This problem is known as Online Knapsack Problem. A solution to this is that, consider that elements are not coming in sequential order, they arrive in a random order [16].

Knapsack problem is well known NP problem. It can't be solved in a polynomial amount of time so there are various approaches for solving 0/1 Knapsack problem such as greedy technique, backtracking, dynamic programming etc.

2.3.2 Analysis of 0/1 knapsack with Dynamic Programming:

0/1 Knapsack problem can be solved with the help of dynamic programming as follows:

Let us consider w_1, w_2, w_3, w_4 ….w_n, w_i's are positive integers, w is weight of knapsack. Let m[i, w] to be the maximum value.

The m[i, w] can define recursively as follows:

m[i, w] = m[i-1 , w] if $w_i > w$ (if the current weight w_i is greater than w)

m[i, w] = max (m[i-1 , w] , m[i-1,w-w_i]+ v_i) $w_i \leq w$

Dynamic programming considers all the possible solution to solve the problem so it is a NP problem.

2.3.3 Analysis of 0/1 Knapsack using Greedy Strategy:

0/1 Knapsack problem can be solved with the help of greedy strategy as follows:

According to the greedy strategy the algorithm takes the solution which appears best at the moment [17]. These algorithms don't always give the optimal solution. Sometimes it will give the solution which is very close to the optimal and sometimes it will give the solution which is very far away from the optimal solution. However unlike dynamic programming it can solve 0/1 knapsack problem in polynomial

amount of time. However these are the simplest and efficient algorithm to solve a problem [18] .

Analysis of Greedy Strategy can be done by the following 3 ways [13]:

1. Greedy Strategy Of Value
2. Greedy Strategy Of Capacity
3. Greedy Strategy Of Unit Value

Greedy Strategy of Value says that firstly choose the item in the knapsack having the largest profit (value) among all the items such that weight of the item wouldn't exceed the volume of the knapsack. Similarly choose the next largest profit item and put it into the knapsack and continue the process. Consider an example to understand Greedy Strategy of Value. Let us consider total no. of items n=3, weight of items w= [20, 10, 18], profit on items p= [15, 10, 10] and the capacity of knapsack A=20. When one is using the above strategy, the solution for this are X = (1, 0, 0), and the total profit for the above problem is 15. But if one sees the above problem the optimal solution should be (0, 1, 1) the total profit should be 20. We can simply see from the above example that this strategy would not guarantee to give optimal solution.

Greedy Strategy of Capacity says that consider the items which have the smallest weight among all the items. Firstly put the item having the smallest weight from the set of items into the knapsack then select the other item from the remaining unselected items and put that into the knapsack and continue the process. Let us consider total no. of items n=2, weight of items w= [10, 20], profit on items p= [5, 100] and the capacity of knapsack A=25. When one is using the above strategy, the solution for this are X = (1, 0), and the total profit for the above problem is 5. But if one sees the above problem the optimal solution should be (0, 1) and the total profit should be 100. One can simply see from the above example that this strategy would not guarantee to give optimal solution [19] .

Greedy Strategy of Unit Value strategy considers both Greedy Strategy of Capacity and Greedy Strategy of Value, and according to this strategy the item whose p_i/w_i value is largest that item puts into the knapsack and the process continues for the remaining items. Let us consider total no. of items n=3, weight of items w= [20, 15, 15], profit on items p= [40, 25, 25] and the capacity of knapsack A=30. When one is

using the above strategy, the solution for this are X = (1, 0, 0), and the total profit for the above problem is 40. But if one sees the above problem the optimal solution should be (0, 1, 1) the total profit should be 65. One can simply see from the above example that this strategy would not guarantee to gets optimal solution also.

The above methods such as Dynamic Programming considers all the possible solution for solving 0/1 knapsack problem and it gives optimal solution but it can't take polynomial time to solve the problem. Similarly Greedy Strategy takes optimal time to solve 0/1 knapsack problem but it can't guarantee to give optimal solution for this. So there is a need of a method to solve this in a polynomial time and gives an optimal solution.

So one can use Genetic Algorithms to solve 0/1 knapsack problem. One can solve 0/1 knapsack problem with genetic algorithm and can get optimal solution. As 0/1 knapsack problem is a NP problem but when we solve this problem with the help of genetic algorithm it will no longer to be a NP problem. With the help of Genetic Algorithm one will get optimal solution. So Genetic Algorithm can be thought as best approach for finding solutions to the problem for which there exists no feasible solution.

2.4 Genetic Algorithms

These algorithms involve in finding solutions to the problems for which no acceptable algorithmic solution exists. The GA methodology involves optimization. In optimization there is a solution space consisting of large number of solution and one have to search one or more very good solutions for our problem. Genetic Algorithms continuously reduce the search space by eliminating the solutions which ranked as poor, and generate a new generation through crossover and mutation those ranked as good. Ranking is based on the Fitness value of each candidate.

2.4.1 How GA Works

Genetic Algorithm is a technique for optimization and also a search technique that is similar to biological evolution. It continuously alters a population of solutions until we don't have an optimal solution.

It starts with random selection of initial population. Then it calculates the fitness value for each individual in the population. Selection is done on the basis of fitness and alteration using crossover and mutation. The selection Crossover and mutation leads to population by eliminating bad solutions and replaces with better solutions. The evolutionary cycle continues until there isn't an acceptable solution or until termination condition [19].

2.4.2 Steps in Typical Genetic Algorithm

1. Start: Randomly generate a population of N chromosomes.
2. Fitness: Calculate the fitness of all chromosomes.
3. Create a new population:
    a. Selection: Select 2 chromosomes from the population according to the selection method.
    b. Crossover: Perform crossover on the selected chromosomes with crossover probability.
    c. Mutation: Perform mutation on the chromosomes obtained after crossover with mutation probability.
4. Replace: Replace the current population with the new population.
5. Test: Test whether the end condition is satisfied. If satisfied then stop and if end condition not satisfied then go to Step 2.

Each iteration of this process is called generation [20-21].

2.4.3 Initialize Population

Initialization of population is the very first step in the genetic algorithm. The search space of Genetic Algorithm consists of population of individual, each individual representing a possible solution to given problem. After initialization the encoding of each individual is done. The encoding of each individual is done as variables or alphabets and mostly binary alphabets {0, 1}.

Size of Population = Number of Chromosomes

2.4.4 Various Encoding methods are

a. Binary Encoding
b. Real Value Encoding
c. Permutation Encoding

d. Tree Encoding

a. Binary Encoding

It is the most common encoding technique. In this encoding each chromosome is represented as a string of 0 or 1.If there are small number of chromosomes then also binary encoding gives possible chromosomes. Also this type of encoding is not suitable for some problems such as for real values binary encoding is not possible. For example, in knapsack problem, we have items each associated with value and size. The capacity of knapsack is also given. We have to select things in such a way that value will be maximized without exceeding the capacity of a knapsack. So with binary encoding if the bit of chromosome is 1 then that item will be included in knapsack if it is 0 then it will be discarded from the knapsack.

b. Real Value Encoding

In real numbers we can't use binary encoding because it will be difficult. So in such type of problems we can use real value encoding. For some special problems real value encoding is very good. This type of encoding is useful in finding weights for neural network.

c. Permutation Encoding

We can use permutation encoding in ordering problems where we can't use binary or real value encoding problems such as travelling salesman problem or task ordering problem we can use this type of encoding. This type of encoding is suitable for such type of problems where we need ordering. We can use this type of encoding in problems like travelling salesman problem, in which permutation encoding shows the sequence of cities means in which order salesman will visit them and distance will be minimized.

d. Tree Encoding

If we want to evolve programs and expressions then we will use this type of encoding. tree encoding can be used in LISP because in this parsing can be done and with the help of tree encoding tree will be parsed easily. Also in tree encoding crossover and mutation can be done easily.

2.4.5 Fitness Function

A fitness function is an objective function. Objective function can be for maximization or it can be for minimization. With the help of fitness function one will calculate the fitness for each chromosome. After calculating fitness for each chromosome, the fitness will be assigned to each chromosome.

2.4.6 Selection

One of the most important operators in genetic algorithms is the selection operator. The performance of algorithm is significantly influenced by this operator. In biological evolution, the genes will be responsible of creation of next generation. Selection process is also based on similar concept. Selection is the process in which individual chromosome is selected from population. After selecting the chromosomes later crossover and mutation will be done. One of the most common forms of selection is fitness proportional selection.

Various selection procedures are:

a. Tournament Selection Method
b. Roulette Wheel (Proportionate Selection)
c. Rank Selection Method
d. Steady State Selection

a. Tournament Selection Method

In this method, two individuals from population are selected for tournament. Tournament is done on the basis of their fitness value. The chromosome having greater fitness value will go to the next generation and become a part of next generation. Advantage of this is that it is very efficient and easy to implement but disadvantage is that one may lose better solution.

b. Roulette Wheel (Proportionate Selection)

The most frequently used selection operator is Roulette wheel selection for the implementation of Genetic Algorithms. However the population diversity of the algorithm and the convergence speed isn't performed with this selection operator [12]. In Roulette Wheel selection, as in all selection methods, the very first step is assigning of fitness by fitness function to solutions or chromosomes. This fitness value is responsible for the selection of each individual chromosome. If f_i is the fitness of

individual i in the population, its probability of being selected is $p_i = f_i / \sum f_j$, $1 \leq j \leq N$,where N is the number of chromosomes.

Now find the cumulative Probability $CP_i = \sum p_j$ $1 \leq j \leq i$.

Then Generate Random Number between 0 and 1. Now choose the chromosome whose cumulative probability just greater than the value generated by random generator, that chromosome will go to the next generation. Advantage is that it is easy to implement and disadvantage of this is that a solution may be copied multiple times. So algorithm relies on mutation operator and also best solution may be discarded from the new generation.

c. Rank Selection Method

There is a problem associated with roulette wheel selection when fitness values differ very much. For example, if there is a best chromosome and its fitness is 90% of all roulette wheels then the remaining chromosome have a little chance to be selected in population. So this problem is solved by rank selection method. The rank selection method fist gives rank on the basis of their fitness. The worst fitness will get rank 1, the second worst will have fitness 2, and the best will have fitness N where N is number of chromosomes in the population. So in this all chromosomes have a chance to get selected. But in this method the convergence is slower because chromosomes do not differ much from others.

d. Steady State Selection

This is not a specific method of selecting parents. In steady state selection some chromosomes for next the population are selected which have high fitness and the remaining are selected randomly.

2.4.7 Crossover

Crossover is the primary search operator in genetic algorithms. There are different types of crossover operators which can be used. Many times it have been proved that uniform crossover can search sometime more efficiently than other crossover operators. Similarly two-point crossover can sometimes search more efficiently than one-point crossover. Also there are no analytical results that states why there are differences in the performance in the crossover operators [22].

Different types of crossover operators are:

a. One Point Crossover
b. Two point Crossover
c. Arithmetic Crossover

a. One Point Crossover

In one point crossover, the crossover operator randomly selects one point that point is called crossover point. Now it will select two parents from the population and interchange the two parent from the crossover point such that after the crossover point genes of parent1 will copied to parent 2 and vice versa and two new children will be produced. Let us consider 2 parents which have been selected for crossover operation. The "|" symbol indicates the crossover point.

Parent1: 11111|111
Parent 2: 10101|000

After applying crossover and interchanging the parent at the crossover point, the following children are produced:

Child1: 11111|000
Child2: 10101|111

b. Two Point Crossover

It is a crossover operator that will select two crossover points within a chromosome and then interchanges the parent1 and parent2 chromosomes between these two points and in this way it produces two new child chromosomes.

For example, let us consider the following two parents and then apply this two point crossover on these two parents. The "|" symbol indicate the randomly chosen crossover points.

Parent 1: 110|111|10
Parent 2: 001|101|11

After interchanging the parent chromosomes between the crossover points, the following child chromosomes are produced.

Child 1: 110|101|10
Child 2: 001|111|11

c. Arithmetic Crossover

It is a crossover operator which uses the following equations to produce two new child chromosomes by linearly combining two parent chromosomes:

Child1 = w * Parent1 + (1 – w) * Parent 2
Child 2 = (1- w) * Parent1 + w * Parent 2

Where w is a random weighting factor. Weighing factor is chosen before crossover operation.

Let us consider the following 2 parents selected for crossover operation. Each chromosome consists of 4 float genes.

Parent 1: (0.3)(0.2)(0.5)(1.5)
Parent 2: (0.5)(0.1)(0.3)(0.8)

If w = 0.3, the following two children would be produced:
Child1:(0.44)(.13)(0.36)(1.01)
Child2:(0.36)(0.17)(0.44)(1.29)

2.4.8 Advanced Crossover Operators In Genetic Algorithm:

There are two advanced crossover operators in GA[23]:

a. Partially matched crossover(PMX)
b. Order Crossover(OX)

a. Partially Matched Crossover (PMX)

PMX is used in travelling salesman like ordering problems. In travelling salesman problem there is a set of cities and a salesman has to visit all the cities to complete a tour in order to minimize the distance. The salesman has to travel all the distance blindly means he is unaware of the distance between the cities. This problem can be seen as normal ordering like problem i.e. Travelling Salesman Problem in permutation form. Let us consider an example of eight cities problem the cities are visited in ascending order to complete a tour can be represented as follows:

1 2 3 4 5 6 7 8

When eight cities visited in descending order to complete a tour then their permutation of cities to complete a tour can be represented as follows:

8 7 6 5 4 3 2 1

In the above method of permutation the fitness will depend upon the ordering value i.e. ascending or descending value. So population gets stagnated in early stages of population. So here Partially Matched Crossover (PMX) will work.

In PMX, there are two strings and also two crossing points are selected randomly. These two points is used to show position - by - position exchange operations. For e.g. consider the following two strings:

A = 1 3 2 | 5 6 7 | 9 8 4 10
B = 9 5 4 | 2 3 10 | 8 7 1 6

PMX always do position wise exchange. PMX firstly do the mapping of String B to String A, firstly the 2 and the 5, the 3 and the 6 and the 10 and the 7 will exchange places. Now similarly the mapping of String A to String B, the 5 and the 2, the 6 and the 3, and the 7 and the 10 will exchange their places. The following two child offspring will produce:

A' = 1 6 5 | 2 3 10 | 9 8 4 7
B' = 9 2 4 | 5 6 7 | 8 10 1 3

Where each child chromosome contains ordered information.

b. Order Crossover

The PMX crossover does point-to-point exchanges of the matching sections in the chromosomes. But instead of using point to point exchanges, Order Crossover (OX) uses sliding motion. The holes which were left by transferring mapped positions the order crossover used sliding motion to fill these holes. For example

A = 1 3 2 | 5 6 7 | 9 8 4 10
B = 9 5 4 | 2 3 10 | 8 7 1 6

When string B mapped to string A like in PMX the cities 5,6,7 leave holes and here H represents the hole in the string.

B = 9 H 4 | 2 3 10 | 8 H 1 H

Now, order crossover uses sliding motion over the above chromosome to fill the holes and after sliding motion the chromosome represented as follows:

B = 2 3 10 H H H 8 1 9 4

Now the above holes i.e. H then filled with the matching section of the mate i.e. matching section of string A. By doing this and completing complimentary cross i.e. same is to be done on string A the following two child offspring will produced.

A' = 5 6 7 | 2 3 10 | 9 8 4 1
B' = 2 3 10 | 5 6 7 | 9 4 8 1

Although PMX and OX are similar, but OX tends to respect relative city position but PMX doesn't. PMX tends to respect absolute city position.

2.4.9 Mutation

Mutation is a random adjustment operator in genetic algorithms. Those characteristics that are not achieved by crossover alone, mutation is useful for introducing that in population [24]. Crossover gives new combinations only by rearranging existing characteristics. For example suppose there are two chromosomes each start with bit 1 when one apply the crossover on these two parents then the new child will produce having first bit 1 so it only rearranges but with the help of mutation the first bit of the child chromosome can be made 0 by flipping the bits 0 bit to a 1 or vice versa. Mutation is also useful for introducing new characteristics in the population.

Various types of mutation operators are:

a. Flip Bit Mutation
b. Boundary Mutation
c. Non Uniform Mutation
d. Uniform Mutation

a. Flip Bit Mutation

This mutation operator inverts the bit of chromosome. If chromosome bit is 1, it changes it to 0 and vice versa.

b. Boundary Mutation

There are two bounds lower bound and upper bound. This mutation operator replaces the chromosome with either upper bound or lower bound and can be used for integer and float genes.

c. Non-Uniform Mutation

Sometimes population gets stagnated in early stages of the evolution. With the help of this mutation operator one can fine tune the solution in later stages of evolution and this can be used for integer and float genes.

d. Uniform Mutation

It randomly selects a uniform random value between the user-specified upper and lower bound for a particular gene and replaces the value of that chosen gene with this random value. This mutation operator can be used for integer and float genes [25] .

2.4.10 Termination

Termination the last step in the genetic algorithms, is the condition according to which genetic algorithm decides whether to continue with the whole procedure again or to stops the search. Termination criteria are checked after each generation to see whether to continue with the search or stop the search.

Various Termination Conditions are

a. Generation Number
b. Evolution Time
c. Fitness Threshold
d. Fitness Convergence
e. Population Convergence

a. Generation Number

In this user defines maximum number of evolutions that has to run. When the user defined evolutions have been run, then this method stops the evolution. This termination method is always active.

b. Evolution Time

In this user defines maximum evolution time**.** When the elapsed evolution time exceeds the user-specified max evolution time then this termination method stops the evolution. By default, until the evolution of the current generation has completed, the evolution is not stopped, but to stop the evolution within a generation this behavior can be changed.

c. Fitness Threshold

In this user will specify a fitness threshold. This termination method stops the evolution until the user defined fitness threshold is achieved. Also this termination method stops the evolution until there will be the best fitness in the population and that fitness will be greater than user defined fitness threshold if our objective is to maximize the fitness.

d. Fitness Convergence

When the fitness is deemed to be converged this termination method will stop the evolution.

e. Population Convergence

When the population is deemed to be converged this termination method will stop the evolution [26].

With the help of Genetic Algorithms, the 0/1 Knapsack problem now no longer remain NP problem as 0/1 knapsack problem now can be solve in a finite amount of time. The results of Genetic Algorithms are totally dependent on operator probability used. Operators used are Selection operator, Crossover operator and Mutation Operator. The Genetic Algorithms can be made more powerful by using adaptive operator probabilities. The quality of the solutions can be increased by adopting adaptive operator probability. In some cases, the operator probability can automatically be adjusted according to quality of solutions [27]. When diversity in the solution space does not take place then solution leads to convergence. Premature Convergence is a major problem in AGA, but with the help of diversity, this problem can be handled [28]. Srinivas introduced modified adaptive crossover strategy with diversity guided mutation. Also an adaptive genetic algorithm with diversity –guided mutation was developed known as MHAGA [30].

2.5 Adaptive Genetic Algorithm (AGA)

The selection, crossover and mutation operator are performed with certain probability. The operator probability in genetic algorithm can be made adaptive according to the situation to produce good solutions. The genetic algorithm with adaptive probability is known as Adaptive Genetic Algorithm (AGA). Although AGA gives good results but it may lead to premature convergence in early stages of genetic algorithm [27]. This means that the population gets stagnated in early stages of algorithm.

2.6 Modified Hybrid Adaptive Genetic Algorithm (MHAGA)

As AGA uses adaptive crossover and mutation still it leads to premature convergence and stagnation at early stages of evolution. So to overcome this, adaptive crossover and mutation were mixed with diversity guided mutation and it led to Modified Hybrid Genetic Algorithm (MHAGA) [30].

2.6.1 Modified Adaptive Operators:

The key factors that affect the performance of genetic algorithms is crossover and mutation. In adaptive crossover and mutation probabilities of individuals (say i and j) denoted as p_c (i , j) and p_m(i) respectively[31]. According to Srinivas the adaptive crossover and mutation probability can be calculated as [27]:

$$p_{c(i,j)} = \begin{cases} k_1 \frac{f_{max} - \acute{f}}{f_{max} - f_{avg}}, & \acute{f} \geq f_{avg} \\ k_2, & \acute{f} < f_{avg} \end{cases} \quad (1)$$

$$p_{m(i)} = \begin{cases} k_3 \frac{f_{max} - f}{f_{max} - f_{avg}}, & f \geq f_{avg} \\ k_4, & f < f_{avg} \end{cases} \quad (2)$$

Fig 2.1: Adaptive Crossover and Mutation Probability

where, $0<k_1$, $k_2<=1$, $0<k_3$, $k_4<1$

f_{avg} = average fitness value of the population

f_{max}= maximum fitness value of the population

$\acute{f}$= largest fitness value among individuals i and j to be crossed

f = fitness value of i to be muted

According to Srinivas modifying formula from (1) and (2) the converge speed may be increase but it has two disadvantages

1. The building block of individual with higher fitness value (best individual) may not be possible because it is taking crossover probability of best individual very small or close to zero.
2. It also leads to premature convergence.

Zong [30] improves the above Srinivas formula as

$$p_{c(\acute{i},j)} = \begin{cases} k_1 \frac{\acute{f} - f_{min}}{f_{avg} - f_{min}}, & \acute{f} < f_{avg} \\ k_2, & \acute{f} \geq f_{avg} \end{cases} \quad (3)$$

Fig 2.2: Zong's Adaptive Crossover Probability[30]

where f_{min} = minimum fitness value of population
$\grave{f}$ = smallest fitness value among individuals i and j to be crossed.

The above formula removes the disadvantages in earlier approach. This approach can produce better individuals. Hence, when applying crossover on the better individuals better children may come. Hence it gives good results.

2.6.2 Comparison of AGA and MHAGA

When solving 0/1 Knapsack with AGA and MHAGA, differences observed are summarized in table 1([27-29]):

Table 2.1: Comparisons of AGA and MHAGA

S.No	Parameter	Adaptive Genetic Algorithm	Modified Hybrid Adaptive Genetic Algorithm
1.	Premature Convergence	Earlier	Later
2.	Crossover Strategies	Adaptive	Adaptive
3.	Mutation Strategies	Adaptive	Diversity Guided
4.	Optimizer	Effective	More Effective
5.	Quality Of Solutions	Good	Best
6.	Accuracy	Low	High
7.	Robustness	Low	High
8.	Stability	Low	High
9.	Time	More	Less
10.	Reliability	Low	High

On comparing AGA and MHAGA with above parameters, it can be concluded that MHAGA proves a better technique for solving 0/1 Knapsack Problem than AGA. As AGA has high premature convergence, the quality of solutions are best obtained in MHAGA than AGA. Also MHAGA is faster than AGA. So MHAGA has proven to be a good technique for solving 0/1 knapsack problem in terms of premature convergence, crossover strategy, mutation strategy, accuracy, reliability etc.

2.7 Experimental Analysis of Genetic Algorithm Based On Greedy Strategy versus Traditional Genetic Algorithm

It is concluded in an experimental analysis that when taking size of knapsack for N=50, and W[50]= (35,34,31, ... 28,27) is the weight and P [50] = (13,18,21 ... ... 25 , 23) is the profit associated with each item. The algorithm runs upto 100 iterations, with crossover probability of 0.15 and mutation probability of 0.05. The experiment runs the traditional genetic algorithm 50 times, with an average of 38 algebra, and obtain an optimal solution for an average of 3382.24.Now the genetic algorithm which is based on greedy strategy use the result of greedy strategy by first initializing code for the population (1101 ... ... .101) with the use of an average 27 algebra, optimal solution for the average 1412.19. Again and again by analyzing optimal solution the genetic algorithm based on greedy strategy proved significantly superior than traditional genetic algorithm [13] .

CHAPTER 3: PROBLEM STATEMENT

3.1 Gaps in Study: NP (non-deterministic polynomial) problems are such problems for which there are no possible algorithms that would guarantee to run the problem in a polynomial time.

The existing methods such as Dynamic Programming considers all the possible solution for solving 0/1 knapsack problem and it gives optimal solution but it can't take polynomial time to solve the problem. Similarly Greedy Strategy takes optimal time to solve 0/1 knapsack problem but it can't guarantee to give optimal solution for this. So there is a need of a method to solve this in a polynomial time and gives an optimal solution.

Table 3.1: Performance of Dynamic versus Greedy Approach

Approach	Optimal Solution	Polynomial Time
Dynamic Programming	Yes	No
Greedy Strategy	Can't guarantee	Yes

3.2 Problem Statement:

Existing GA algorithm does not provide optimal solution.

3.2.1 Existing Algorithm

The existing algorithm for solving 0/1 Knapsack problem with GA is given below:

1. Start: Randomly generate a population of N chromosomes.
2. Fitness: Calculate the fitness of all chromosomes.
3. Create a new population:
    a. Selection: Select 2 chromosomes from the population according to the selection method.
    b. Crossover: Perform crossover on the selected chromosomes with crossover probability.
    c. Mutation: Perform mutation on the chromosomes obtained after crossover with mutation probability.
4. Replace: Replace the current population with the new population.

5. Test: Test whether the end condition is satisfied. If satisfied then stop and if end condition not satisfied then go to Step 2.

Each iteration of this process is called generation [20] [21] [22].

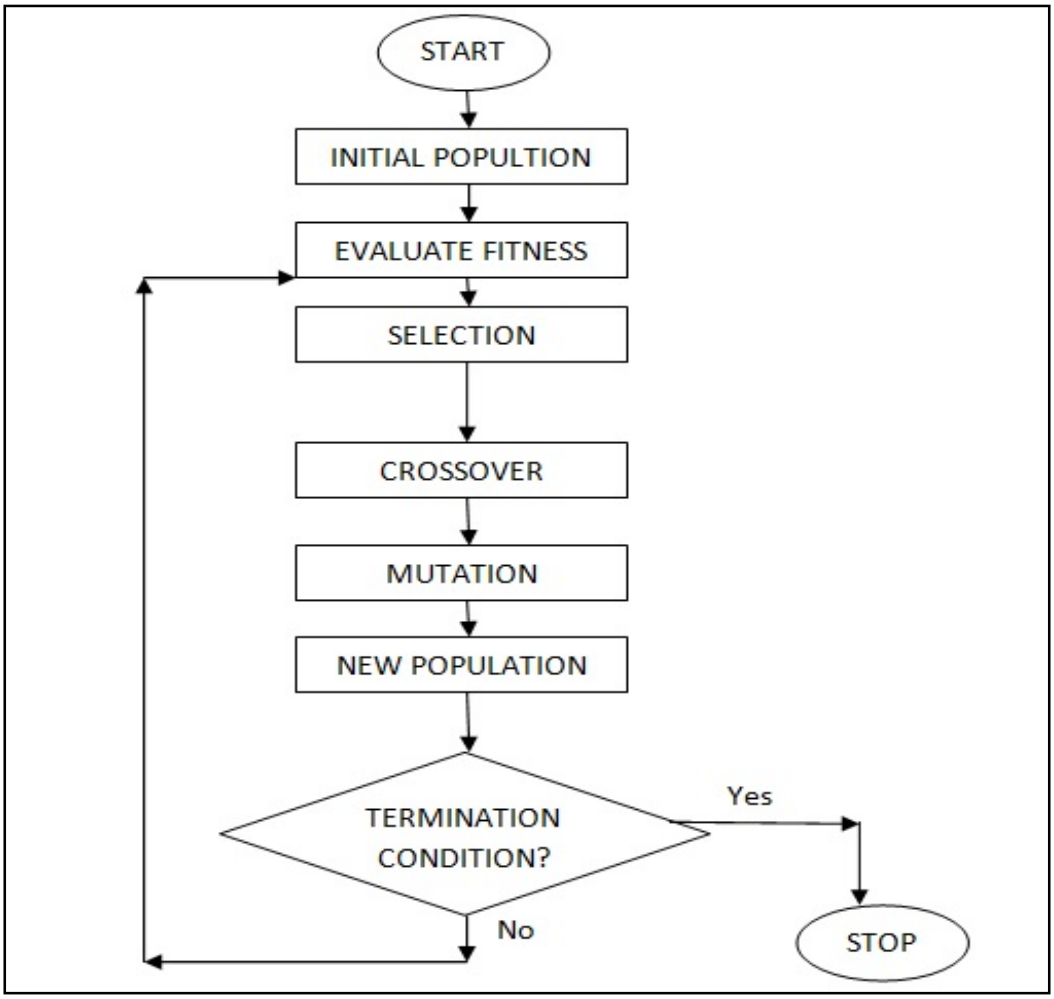


Fig 3.1: Flowchart of Existing Algorithm [31]

3.2.2 Problem in Existing Algorithm

In the existing approach it is randomly selecting the chromosomes from the population and after selecting the chromosomes, it is performing crossover and mutation on the selected chromosomes. But it is possible that after performing crossover and mutation the fitness of the chromosome will degrade and then it will those altered chromosomes to new population and in the end it will blindly replace new population with the older population. So in this way the performance of this algorithm is not good because some better solutions may be discarded from the population.

3.2.3 Objectives of Proposed Research Work:

1. In order to eliminate, the problem in existing Genetic Algorithm, a new improved algorithm is required which can help to prevent in discarding good solutions from the intermediate population.
2. The algorithm should also give optimal result i.e. solution should be obtained in earlier generations with better fitness value

CHAPTER4: IMPLEMENTATION

4.1 Proposed Algorithm

In order to improve the performance of the existing algorithm new algorithm is proposed, in the way that it can create an intermediate population. Now proposed algorithm compares the i^{th} chromosome of the intermediate population with the old population and after comparing the chromosomes whose fitness value from the both population is better than that will be a part of new generation.

1. Start: Randomly generate a population of N chromosomes.
2. Fitness: Calculate the fitness of all chromosomes.
3. Create an intermediate population:
    a) Selection: Select 2 chromosomes from the population according to the selection method.
    b) Crossover: Perform crossover on the selected chromosomes with crossover probability.
    c) Mutation: Perform mutation on the chromosomes obtained after crossover with mutation probability.
4. Create a new population:

Compare the i^{th} chromosome of initial population and intermediate population whichever from both the population have better fitness value that will be a part of new generation.

5. Replace: Replace the current population with the new population.
6. Test: Test whether the end condition is satisfied. If satisfied then stop and if end condition not satisfied then go to Step 2.

The proposed algorithm can obtain better results in the earlier generations and as the generations increases the fitness will getting better from its earlier generation. Also variations in later generation will be smaller. The implementation and results of proposed algorithm are discussed in detail in subsequent sections.

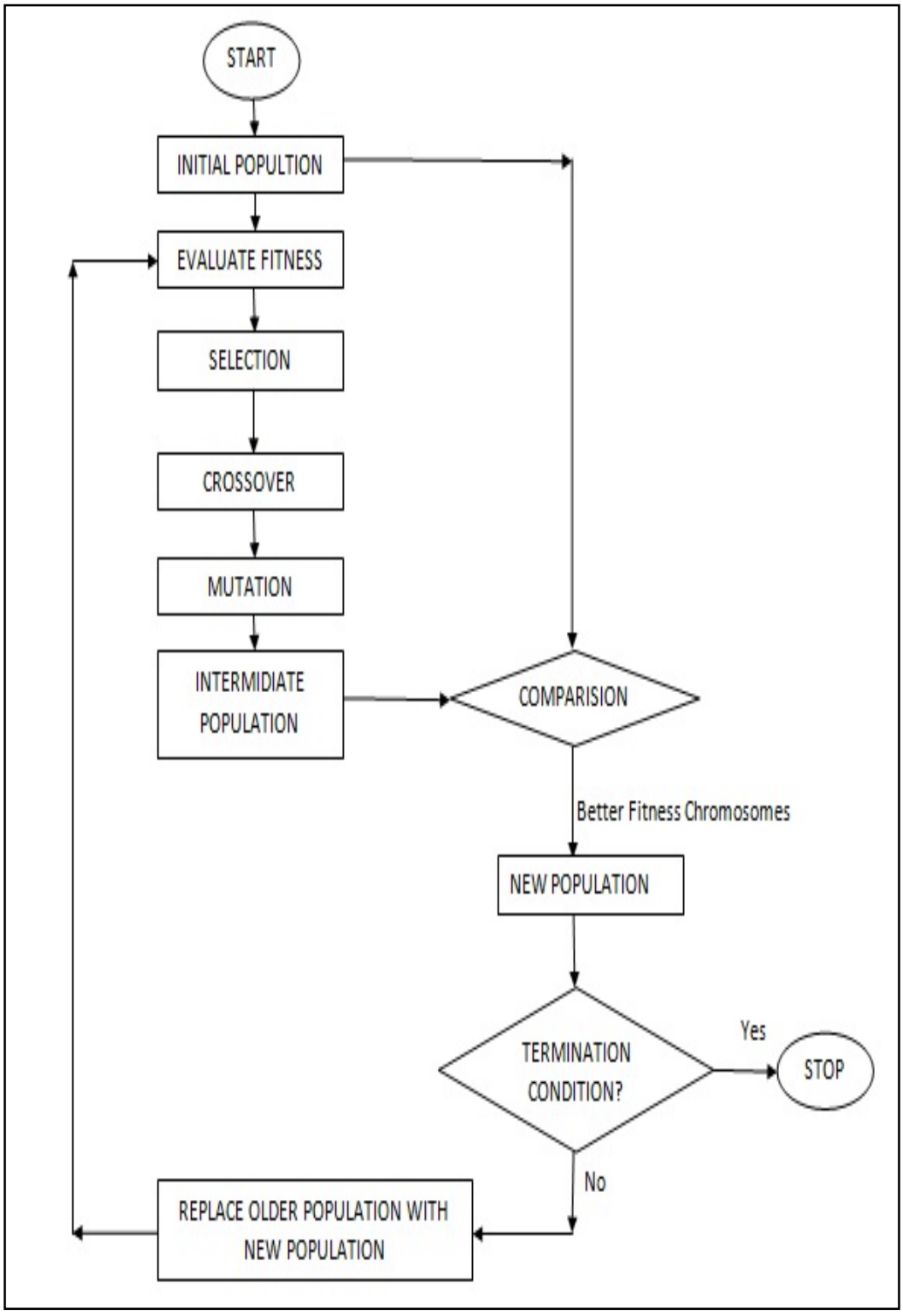


Fig 4.1: Flowchart of Proposed Algorithm

4.2 Platform Description

The algorithm has been implemented in Java. Java is an open source software and from 2006 it become easily available under GNU General Public License (GPL) and therefore is called as Openjdk. Java is an object oriented language and it uses JVM (java virtual machine) to execute programs. The programmer first creates a java file and then that java file is compiled using java compiler. After compiling the file java compiler creates a bytecode of that file. This bytecode is then interpreted using java virtual machine (JVM)[32].

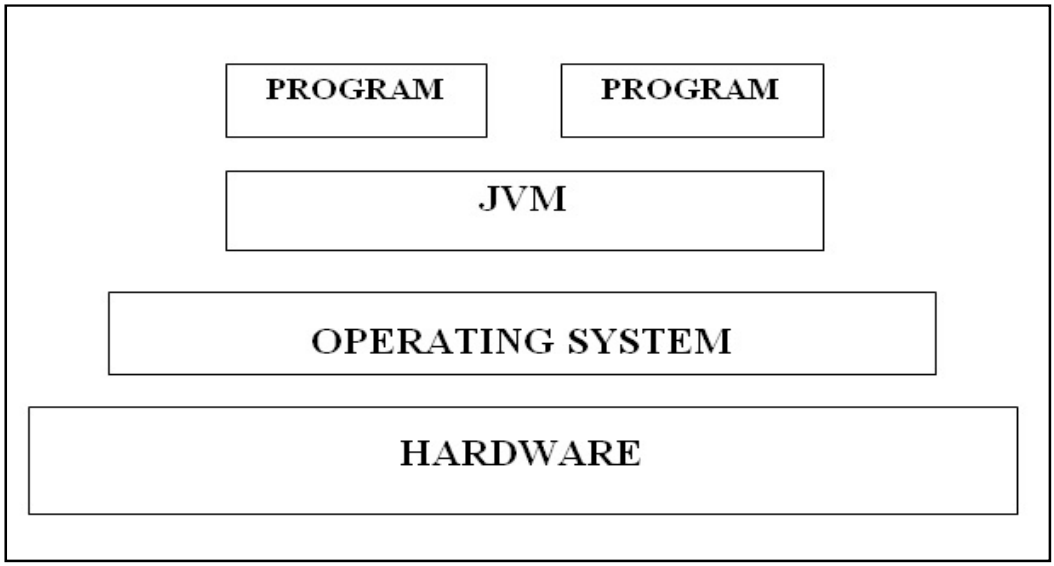


Fig 4.2: Java Virtual Machine (JVM) [32]

4.2.1 Features of Java [32] [33]

- Java is a simple language i.e. it is easy to write, easily understandable, eye catching, easily readable, easy to learn and use.
- Java is a secure language. With the help of java one can develop secure internet applications. Also with the help of java one can securely access web applications.
- Java is Portable i.e. the applications which are created on one platform can be easily executed on other platform.

- Java is a Object-oriented programming language. C++ is semi object oriented language but java is pure object oriented language.
- With performing run-time checks one can easily make error-free programs, this makes java robust.
- With the help of java dynamic web pages can be easily created.
- Java is a high performance language. Java Virtual Machine executed bytecode faster as bytecodes are highly optimized.
- Java is interpreted as well as compiled language.
- Java automatically do memory management. Garbage collector is there in java which deallocates memory from the objects to which no active pointer to memory exists.
- Java is an architecture-neutral language. It is a machine independent language. It means java is independent of hardware. Java is not related to specific operating system architecture.
- Java is a distributed language. Java can transmit or run over internet.

Also the algorithm is implemented using Net Beans as Run time environment. The version used is netbeans-6.5. It is an Integrated Development Environment (IDE). With the help of Net Beans, one can easily create applications in java as well as in other languages also. Also one can make a jar file of the project. Project directly runs by clicking on this jar file.

4.2.2 Why to use Net Beans IDE [34]

- It is freely available
- It is an Open Source
- Customizable and extensible
- It Supports java standards and other platforms as well
- Powerful
- Debugging tools
- Award winning GUI Builder

Let us consider a real life application of 0/1 knapsack problem and application is related to our day to day life where one have to purchase things and has constraints

with us i.e. within given constraints of money one have to manage our budget economically. The application includes a buyer, who has to invest his money for purchasing items. Every item is associated with a price and also has a margin. The margin indicates the profit which the buyer gets when he sell those items.

This is similar to knapsack problem where weight of knapsack and items with weights and their associated profit value are given. By relating this application with knapsack, weight of bag is considered as total money that the buyer has and the items weights related to item price and the value in knapsack is related to the profit margin.

4.3 Encoding of Chromosomes

Binary Encoding is used to implement proposed algorithm. In Binary Encoding chromosome is represented in the sequence of 0 and 1, where 1 indicates that the item is included in knapsack and 0 indicates that item is not included in knapsack.

4.4 Generation of Chromosomes

Generation of chromosome is done to implement the algorithm with the help of generate_Chromosome() function. It will take list of item_price , population size , total money as input to the function and the function will return CHROMOSOME[][] matrix in which each row consists of one possible solution. CHROMOSOME [][] matrix having dimensions population size and m where m is total number of items.

```
procedure generate_Chromosome(item_price, population, tot_money)
 1  for i=1 to population
 2     w=0
 3     for j=1 to m
 4        b=round(1+m*random)
 5        if(b>m)
 6           b=b-1
 7        if(CHROMOSOME[i][b])==0 &&(w+item_price[b])<=tot_money)
 8           CHROMOSOME[i][b] =1
 9           w=w+item_price[b]
10  return CHROMOSOME
```

Fig 4.3: Algorithm for Generation of Chromosomes

After generation of chromosomes fitness of chromosomes will be calculated.

4.5 Calculation of Fitness

Fitness of the chromosome is calculated by function fitness_of_cromosomes which will take CHROMOSOME [][] , margin[], a global array Fitness to initialize the fitness value for each chromosome. The function will return Fitness [] as fitness value for each chromosome in the population.

```
procedure fitness_of_cromosomes(CHROMOSOME, margin, Fitness)
1  for i=1to population size
2     f=0
3     for j=1 to no. of items
4        f=f+(margin[j]*CHROMOSOME[i][j]);
5     Fitness[i] =f
6  return Fitness
```

Fig 4.4: Algorithm for Calculation of Fitness

According to fitness value, solutions may be accepted or rejected for next generation.

4.6 Generation of Intermediate Population:

4.6.1 Selection of Chromosomes

The algorithm uses tournament selection method where it randomly select four chromosomes from the population and then compare first two chromosomes. The chromosome with greater fitness value will go to the next generation for further iteration and then similarly do this for last two chromosomes.

```
procedure selection_of_chromosome(total_item, Fitness)
 1  for k=1 to 4
 2      d=round(1+rand()*total_item)
 3      if(d>total_item)
 4          d=d-1
 5          result[k] =round[d]
 6    parent1=0
 7    parent2=0
 8    if(Fitness(result[1])>Fitness(result[2]))
 9       parent1=result[1]
10    else
11       parent1=result[2]
12    if(Fitness(result[3])>Fitness(result[4]))
13       parent2=result[3]
14    else
15       parent2=result[4]
```

Fig 4.5: Algorithm for Selection of Chromosomes Using Tournament Selection

4.6.2 Crossover of Selected Parents

In above selection process it will select two parent chromosomes. Now crossover will be performed on those chromosomes and then child chromosome will be created. The algorithm is using two point crossover here with crossover probability 0.9. Two random points are selected i.e. br1 and br2 for two point crossover. Here parent11 [] is the 1st parent chromosome and parent21[] is the 2nd parent chromosome. The function will generate two child chromosomes.

```
procedure crossover_of_chromosome(parent11,parent21,br1, br2)
 1 child1=parent11
 2 child2=parent21
 3 if(br1<br2)
 4    for b=br1 to br2
 5     child1[b] =parent21[b]
 6     child2[b] =parent11[b]
```

```
7 if(br2<br1)
 8   for b=br2 to br1
 9      child1[b] =parent21[b]
10      child2[b] =parent11[b]
```

Fig 4.6: Algorithm for Crossover Using 2 Point Crossover

After this operation there will be two child chromosomes .Now the total price of child chromosomes will be calculated, The total price of child chromosome can be calculated with the help of computePrice function this function will take child chromosome and item_price[] as input. If its total value will exceed the total money buyer has then child chromosome will be discarded and parent is considered for next generation.

```
procedure computePrice(ch1, item_price)
1 for i=1 to n
2     price=price+(ch1[i]*item_price[i])
3 return price
```

Fig 4.7: Algorithm Showing Computation of Price for Chromosome

4.6.3 Mutation Of Chromosome

Project uses Flip bit mutation with mutation probability 1/n where n is total items in population. With the help of crossover and mutation there is a possibility for better solution. The function mutation_of_chromsome will perform mutation. The function will take child1 and child2 and one break point as input and performs mutation on the child process by flipping the bit in chromosome i.e. 1 to 0 and 0 to 1.

```
procedure mutation_of_chromosome(ch1, ch2, break1)
 1 Child1=ch1
 2 Child2=ch2
 3 for i=break1 to s,
 4     if(Child1[i]==0)
```

```
5       Child1[i] =1
6    else
7       Child1[i] =0
8    if(Child2[i]==0)
9       Child2[i] =1
10   else
11      Child2[i] =0
```

Fig 4.8: Algorithm Showing Mutation of Chromosomes

After mutation again calculate the price of child process with computePrice and test whether price exceeds the total money the buyer have or not. If it exceeds then parent will be selected.

4.7 Generation of New Population

4.7.1 Comparison of Old Population and Intermediate Population

Now at this point there is two matrix CHROMOSOME[][] and CHROMOSOME1[][]. CHROMOSOME[][] matrix will have chromosome before crossover and mutation process. And CHROMOSOME1[][] will have alterated chromosomes i.e. chromosome after crossover and mutation process. Now algorithm have to compare the chromosome in each matrix and have a final CHROMOSOME[][] matrix which contain the chromosomes having higher price between two matrices.

```
procedure compareCromosome(CHROMOSOME,CHROMOSOME1,item_price,
total_money)
1 for i=0 to population size
2   W1 =0
3   W2 =0
4   for j =0 to total items
5       W1=W1 + (CHROMOSOME[i][j] * item_price[j])
6       W2=W2 + (CHROMOSOME1[i][j] *item_price[j])
7    if (W2 > W1 && W2 < total_money)
8    CHROMOSOME[i] = CHROMOSOME1[i]
```

Fig 4.9: Comparison of Old Population and Intermediate Population

4.8 Compute Best Solution in Population

After comparisons of chromosomes now there is a need to calculate the best solution in the population. Best solution can be found with the help of BestSoln function. It takes whole CHROMOSOME[][] matrix ,Fitness[], item_price[] and a flag variable as input and the function calculate the index of the chromosome having maximum fitness. And also it will calculate total price across that index.

```
public void BestSoln(CHROMOSOME, Fitness ,item_price, flag)
 1 max=NEGATIVE_INFINITY
 2 min =POSITIVE_INFINITY
 3 best_prince =0
 4 maxIndex= 0
 5 chromv=0
 6 for i = 0 to population size
 7    if (max < Fitness[i])
 8          max =Fitness[i]
 9          maxIndex =i
10 for (i=0 to 1)
11     for (j=0 to no. of items)
12           chromv = CHROMOSOME[maxIndex][j]
13           best_prince =best_price + chromv * item_price[j]
```

Fig 4.10: Algorithm computing Best Chromosome

4.9 Termination Condition

The project will run up to user's inputted number of generations i.e. generation 15 as soon as the 15^{th} generation will come it will stop and get terminate.

With the above implementation one can find an optimal solution for our list of items to get maximum profit.

CHAPTER 5: EXPERIMENTAL RESULTS

5.1 Test Cases for the Experiment

The experiment has been performed using the following test case [35]:

Table 5.1: Different Test Cases

Test Case	No. of Items	Total Money (Weight of Knapsack)	Item Price (Item Weight)	Item Margin (Item Profit)
1	8	104	25, 35, 45, 5, 25, 3, 2, 2	350, 400, 450, 20, 70, 8, 5, 5
2	10	165	23, 31, 29, 44, 53, 38, 63, 85, 89, 82	92, 57, 49, 68, 60, 43, 67, 84, 87, 42
3	7	170	41, 50, 49, 59, 55, 57, 60	442, 525, 511, 593, 546, 564, 617
4	15	750	70, 73, 77, 80, 82, 87, 90, 94, 98, 106, 110, 113, 115, 118, 120	135, 139, 149, 150, 156, 163, 173, 184, 192, 201, 210, 214, 221, 229, 240
5	5	26	12,7,11,8,9	24,13,23,15,16

5.1.1 Test Case 1:

Total number of items: 8

Total Money the buyer has (can be consider as weight of Knapsack): 104

Items Price (can be consider as item weights): 25, 35, 45, 5, 25, 3, 2, 2

Items Margin (can be consider as profit): 350, 400, 450, 20, 70, 8, 5, 5

So from the above description Item Description is given below:

Table 5.2: Item Description

Item	1	2	3	4	5	6	7	8
Margin	350	400	450	20	70	8	5	5
Price	25	35	45	5	25	3	2	2

Binary Encoding is used here. In Binary Encoding Chromosome is represented in form of sequence of 0 and 1, where 1 indicates that the item is included in knapsack and 0 indicates that item is not included in knapsack. Table II shows how encoding is done.

Table 5.3: Binary Encoding

Item	1	2	3	4	5	6	7	8
Chromosome	0	0	0	1	1	0	1	0
Included?	no	no	no	yes	yes	no	yes	no

Suppose the initial population of chromosomes is given below:

Table 5.4: Initial Population

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7 8	
1	**0 1 0 0 1 1 1 1**	**488**
2	0 1 0 0 1 0 1 1	480
3	0 0 1 0 1 0 0 0	520
4	0 1 1 0 1 0 0 1	925
5	**0 1 0 1 1 1 0 1**	**503**
6	0 1 0 1 1 1 0 1	503
7	0 1 1 0 1 0 0 1	925
8	**0 0 1 0 1 0 0 1**	**525**

Now population after performing selection, crossover and mutation i.e. intermediate population is given below in table 5.5:

Table 5.5: Intermediate Population

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7 8	
1	**0 1 0 0 1 0 1 1**	**480**
2	0 1 0 0 1 0 1 1	480
3	0 0 1 0 1 0 0 0	520
4	0 1 1 0 1 0 0 1	925
5	**0 1 1 0 1 1 0 1**	**933**
6	0 1 0 1 1 1 0 1	503
7	0 1 1 0 1 0 0 1	925
8	**0 1 1 0 1 0 0 1**	**925**

Now new population is created by comparing the above two population, population which have maximum profit value will be a part of newer population as shown in table 5.6.

Table 5.6: New Population

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7 8	
1	**0 1 0 0 1 1 1 1**	**488**
2	0 1 0 0 1 0 1 1	480
3	0 0 1 0 1 0 0 0	520
4	0 1 1 0 1 0 0 1	925
5	**0 1 1 0 1 1 0 1**	**933**
6	0 1 0 1 1 1 0 1	503
7	0 1 1 0 1 0 0 1	925
8	**0 1 1 0 1 0 0 1**	**925**

So in this way the proposed approach selects the best chromosome among the old and initial population.

Following tables 5.7 to 5.13 shows the output of different generations.

Table 5.7: Output of 1st Generation of Test Case 1

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7 8	
1	0 1 0 0 1 1 1 1	488
2	0 1 0 0 1 0 1 1	480
3	0 0 1 0 1 0 0 0	520
4	0 1 1 0 1 0 0 1	925
5	0 1 0 1 1 1 0 1	503
6	0 1 0 1 1 1 0 1	503
7	0 1 1 0 1 0 0 1	925
8	0 0 1 0 1 0 0 1	525

Table 5.8: Output of 2nd Generation of Test Case 1

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7 8	
1	0 1 0 0 1 1 1 1	488
2	0 1 0 0 1 0 1 1	480
3	0 0 1 0 1 0 0 0	520
4	0 1 1 0 1 0 0 1	925
5	0 1 1 0 1 1 0 1	933
6	0 1 0 1 1 1 0 1	503
7	0 1 1 0 1 0 0 1	925
8	0 1 1 0 1 0 0 1	925

Table 5.9: Output of 3rd Generation of Test Case 1

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7 8	
1	0 1 0 0 1 1 1 1	488
2	0 1 0 0 1 0 1 1	480
3	0 1 1 0 1 0 0 0	920
4	0 1 1 0 1 0 0 1	925
5	0 1 1 0 1 1 0 1	933
6	0 1 1 0 1 1 0 1	933
7	0 1 1 0 1 1 0 1	933
8	0 1 1 0 1 1 0 1	933

Table 5.10: Output of 4th Generation of Test Case 1

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7 8	
1	0 1 0 0 1 1 1 1	488
2	0 1 0 0 1 0 1 1	480
3	0 1 1 0 1 0 0 0	920
4	0 1 1 0 1 1 0 1	933
5	0 1 1 0 1 1 0 1	933
6	0 1 1 0 1 1 0 1	933
7	0 1 1 0 1 1 0 1	933
8	0 1 1 0 1 1 0 1	933

Table 5.11: Output of 5th Generation of Test Case 1

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7 8	
1	0 1 0 0 1 1 1 1	488
2	0 1 0 0 1 0 1 1	480
3	0 1 1 0 1 0 0 0	920
4	0 1 1 0 1 1 0 1	933
5	0 1 1 0 1 1 0 1	933
6	0 1 1 0 1 1 0 1	933

7	0 1 1 0 1 1 0 1	933
8	0 1 1 0 1 1 0 1	933

Table 5.12: Output of 6th Generation of Test Case 1

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7 8	
1	0 1 0 0 1 1 1 1	488
2	0 1 0 0 1 0 1 1	480
3	0 1 1 0 1 0 0 0	920
4	0 1 1 0 1 1 0 1	933
5	0 1 1 0 1 1 0 1	933
6	0 1 1 0 1 1 0 1	933
7	0 1 1 0 1 1 0 1	933
8	0 1 1 0 1 1 0 1	933

Similarly the output upto 15th generation is same as generation 5. No change in chromosomes after Generation 5.

Table 5.13: Output of 15th Generation of Test Case 1

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7 8	
1	0 1 0 0 1 1 1 1	488
2	0 1 0 0 1 0 1 1	480
3	0 1 1 0 1 0 0 0	920
4	0 1 1 0 1 1 0 1	933
5	0 1 1 0 1 1 0 1	933
6	0 1 1 0 1 1 0 1	933
7	0 1 1 0 1 1 0 1	933
8	0 1 1 0 1 1 0 1	933

Best Result: Profit Margin 933 and optimal solution is shown in table 5.14.

Table 5.14: Optimal Solution of Test Case 1

Item	1	2	3	4	5	6	7	8
Chromosome	0	1	1	0	1	1	0	1
Included?	no	yes	yes	yes	no	no	yes	yes

So by choosing item 2, 3, 5, 6, 8 and spending Rs 110 the buyer will get a margin of Rs 933 as profit when buyer sells those items and this is the optimal solution.

5.1.2 Test Case 2:

Total number of items: 10

Total Money the buyer has (can be consider as weight of Knapsack): 165

Items Price (can be consider as item weights): 23, 31, 29, 44, 53, 38, 63, 85, 89, 82

Items Margin (can be consider as profit): 92, 57, 49, 68, 60, 43, 67, 84, 87, 42

Population Size: 8

Table 5.15: Output of Generation1 of Test Case 2

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7 8 9 10	
1	0 1 0 1 0 0 0 0 1 0	212
2	0 1 1 0 1 0 0 0 0 0	166
3	0 1 0 0 0 1 0 0 0 1	142
4	1 0 0 0 1 0 0 0 0 1	194
5	0 0 1 1 0 0 0 0 0 1	159
6	0 0 0 0 1 0 0 0 1 0	147
7	0 1 1 0 1 0 0 0 0 0	166
8	0 1 1 0 1 0 0 0 0 0	166

Table 5.16: Output of Generation 2 of Test Case 2

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7 8 9 10	
1	0 1 0 1 0 0 0 0 1 0	212
2	0 1 1 0 1 0 0 0 0 0	166
3	0 1 0 0 1 0 0 0 0 1	159
4	1 0 0 0 1 0 0 0 0 1	194
5	0 0 1 1 0 0 0 0 0 1	159
6	0 1 1 0 1 0 0 0 1 0	253
7	0 1 1 0 1 0 0 0 0 0	166
8	0 1 1 0 1 0 0 0 0 0	166

Table 5.17: Output of Generation 3 of Test Case 2

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7 8 9 10	
1	0 1 0 1 0 0 0 0 1 0	212
2	0 1 1 0 1 0 0 0 0 0	166
3	0 1 0 0 1 0 0 0 0 1	159
4	1 0 0 0 1 0 0 0 0 1	194
5	0 0 1 1 0 0 0 0 0 1	159
6	0 1 1 0 1 0 0 0 1 0	253
7	0 1 1 0 1 0 0 0 0 0	166
8	0 1 1 0 1 0 0 0 0 0	166

Similarly the output upto 15th generation is same as generation 5. No change in chromosomes after Generation 3.

Table 5.18: Output of Generation 15 of Test Case 2

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7 8 9 10	
1	0 1 0 1 0 0 0 0 1 0	212
2	0 1 1 0 1 0 0 0 0 0	166
3	0 1 0 0 1 0 0 0 0 1	159
4	1 0 0 0 1 0 0 0 0 1	194
5	0 0 1 1 0 0 0 0 0 1	159
6	0 1 1 0 1 0 0 0 1 0	253
7	0 1 1 0 1 0 0 0 0 0	166
8	0 1 1 0 1 0 0 0 0 0	166

Best Result: Profit Margin: 253 and optimal solution is shown in table 5.19

Table 5.19: Optimal Solution of Test Case 2

Item	1	2	3	4	5	6	7	8	9	10
Chromosome	0	1	1	0	1	0	0	0	1	0
Included?	no	yes	yes	no	yes	no	no	no	yes	no

5.1.3 Test Case 3:

Total number of items: 7

Total Money the buyer has (can be consider as weight of Knapsack): 170

Items Price (can be consider as item weights): 41, 50, 49, 59, 55, 57, 60

Items Margin (can be consider as profit): 442, 525, 511, 593, 546, 564, 617

Population Size: 6

Following tables 5.20 to 5.25 shows the output of different generations.

Table 5.20: Output of 1st Generation of Test Case 3

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7	
1	1 1 1 0 0 0 0	1478
2	0 1 0 0 0 0 1	1142
3	0 1 0 0 0 0 0	525
4	1 0 0 0 0 0 1	1059
5	1 0 0 0 0 0 1	1059
6	0 1 0 0 0 0 1	1142

Table 5.21: Output of 2nd Generation of Test Case 3

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7	
1	1 1 1 0 0 0 0	1478
2	0 1 0 0 0 0 1	1142
3	0 1 0 1 0 0 0	1118
4	1 0 0 0 0 0 1	1059
5	1 0 0 0 0 0 1	1059
6	0 1 0 0 0 0 1	1142

Table 5.22: Output of 3rd Generation of Test Case 3

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7	
1	1 1 1 0 0 0 0	1478
2	1 1 1 0 0 0 0	1478
3	0 1 0 1 0 0 0	1118
4	1 0 0 0 0 0 1	1059
5	1 0 0 0 0 0 1	1059
6	0 1 0 0 0 0 1	1142

Table 5.23: Output of 4th Generation of Test Case 3

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7	
1	1 1 1 0 0 0 0	1478
2	1 1 1 0 0 0 0	1478
3	0 1 0 1 0 0 0	1118
4	1 0 0 0 0 0 1	1059
5	0 1 0 0 0 1 1	1706
6	0 1 0 0 0 0 1	1142

Table 5.24: Output of 5th Generation of Test Case 3

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7	
1	1 1 1 0 0 0 0	1478
2	1 1 1 0 0 0 0	1478
3	0 1 0 1 0 0 0	1118
4	1 0 0 0 0 0 1	1059
5	0 1 0 0 0 1 1	1706
6	0 1 0 1 0 0 1	1735

Similarly the output upto 15th generation is same as generation 5. No change in chromosomes after Generation 5.

Table 5.25: Output of 15th Generation of Test Case 3

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7	
1	1 1 1 0 0 0 0	1478
2	1 1 1 0 0 0 0	1478
3	0 1 0 1 0 0 0	1118
4	1 0 0 0 0 0 1	1059
5	0 1 0 0 0 1 1	1706
6	0 1 0 1 0 0 1	1735

Best Result: Profit margin: 1735 and optimal solution is shown in table 5.26.

Table 5.26: Optimal Solution of Test Case 3

Item	1	2	3	4	5	6	7
Chromosome	0	1	0	1	0	0	1
Included?	no	yes	no	yes	no	no	yes

5.1.4 Test Case 4:

Total number of items: 15

Total Money the buyer has (can be consider as weight of Knapsack): 750

Items Price (can be consider as item weights): 70, 73, 77, 80, 82, 87, 90, 94, 98, 106, 110, 113, 115, 118, 120

Items Margin (can be consider as profit): 135, 139, 149, 150, 156, 163, 173, 184, 192, 201, 210, 214, 221, 229, 240

Population size: 8

Table 5.27: Output of Generation 1 of Test Case 4

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	
1	0 0 1 0 1 0 0 1 0 1 1 0 0 1 1	1369
2	0 1 0 1 0 1 1 0 0 1 0 1 0 0 1	1280
3	0 1 1 1 1 0 1 0 0 1 0 0 0 1 1	1437
4	0 0 1 0 0 1 1 1 0 0 0 1 1 0 1	1344
5	1 0 1 1 0 1 1 0 1 0 0 0 1 1 0	1412
6	0 1 1 0 0 0 0 0 0 1 1 0 1 1 1	1389
7	0 1 1 0 1 1 1 0 0 1 0 0 1 0 1	1442
8	1 0 0 0 0 0 1 1 0 0 1 1 1 0 1	1377

Table 5.28: Output of Generation 2 of Test Case 4

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	
1	0 0 1 0 1 0 0 1 0 1 1 0 0 1 1	1369
2	0 1 0 1 0 1 1 0 0 1 0 1 0 0 1	1280
3	0 1 1 1 1 0 1 0 0 1 0 0 0 1 1	1437
4	0 0 1 0 0 1 1 1 0 0 0 1 1 0 1	1344
5	1 0 1 1 0 1 1 0 1 0 0 0 1 1 0	1412
6	0 1 1 0 0 0 0 0 0 1 1 0 1 1 1	1389
7	0 1 1 0 1 1 1 0 0 1 0 0 1 0 1	1442
8	1 0 0 0 0 0 1 1 0 0 1 1 1 0 1	1377

Table 5.29: Output of Generation 3 of Test Case 4

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	
1	0 0 1 0 1 0 0 1 0 1 1 0 0 1 1	1369
2	0 1 0 1 0 1 1 0 0 1 0 1 0 0 1	1280
3	0 1 1 1 1 0 1 0 0 1 0 0 0 1 1	1437
4	0 0 1 0 0 1 1 1 0 0 0 1 1 0 1	1344
5	1 0 1 1 0 1 1 0 1 0 0 0 1 1 0	1412
6	0 1 1 0 0 0 0 0 0 1 1 0 1 1 1	1389
7	0 1 1 0 1 1 1 0 0 1 0 0 1 0 1	1442
8	1 0 0 0 0 0 1 1 0 0 1 1 1 0 1	1377

Table 5.30: Output of Generation 4 of Test Case 4

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	
1	0 0 1 0 1 0 0 1 0 1 1 0 0 1 1	1369
2	0 1 0 1 0 1 1 0 0 1 0 1 0 0 1	1280
3	0 1 1 1 1 0 1 0 0 1 0 0 0 1 1	1437
4	0 0 1 0 0 1 1 1 0 0 0 1 1 0 1	1344
5	1 0 1 1 0 1 1 0 1 0 0 0 1 1 0	1412
6	0 1 1 0 0 0 0 0 0 1 1 0 1 1 1	1389
7	0 1 1 0 1 1 1 0 0 1 0 0 1 0 1	1442
8	1 0 1 1 0 1 1 0 1 0 0 0 1 1 0	1412

Table 5.31: Output of Generation 5 of Test Case 4

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	
1	0 0 1 0 1 0 0 1 0 1 1 0 0 1 1	1369
2	0 1 0 1 0 1 1 0 0 1 0 1 0 0 1	1280
3	0 1 1 1 1 0 1 0 0 1 0 0 0 1 1	1437
4	0 0 1 0 0 1 1 1 0 0 0 1 1 0 1	1344
5	1 0 1 1 0 1 1 0 1 0 0 0 1 1 0	1412
6	0 1 1 0 0 0 0 0 0 1 1 0 1 1 1	1389

7	0 1 1 0 1 1 1 0 0 1 0 0 1 0 1	1442
8	1 0 1 1 0 1 1 0 1 0 0 0 1 1 0	1412

Table 5.32: Output of Generation 6 of Test Case 4

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	
1	0 0 1 0 1 0 0 1 0 1 1 0 0 1 1	1369
2	0 1 0 1 0 1 1 0 0 1 0 1 0 0 1	1280
3	0 1 1 1 1 0 1 0 0 1 0 0 0 1 1	1437
4	0 0 1 0 0 1 1 1 0 0 0 1 1 0 1	1344
5	1 0 1 1 0 1 1 0 1 0 0 0 1 1 0	1412
6	0 1 1 0 0 0 0 0 0 1 1 0 1 1 1	1389
7	0 1 1 0 1 1 1 0 0 1 0 0 1 0 1	1442
8	0 1 1 1 1 0 1 0 0 1 0 0 0 1 1	1437

Similarly the output upto 15th generation is same as generation 5. No change in chromosomes after Generation 6.

Table 5.33: Output of Generation 15 of Test Case 4

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	
1	0 0 1 0 1 0 0 1 0 1 1 0 0 1 1	1369
2	0 1 0 1 0 1 1 0 0 1 0 1 0 0 1	1280
3	0 1 1 1 1 0 1 0 0 1 0 0 0 1 1	1437
4	0 0 1 0 0 1 1 1 0 0 0 1 1 0 1	1344
5	1 0 1 1 0 1 1 0 1 0 0 0 1 1 0	1412
6	0 1 1 0 0 0 0 0 0 1 1 0 1 1 1	1389
7	0 1 1 0 1 1 1 0 0 1 0 0 1 0 1	1442
8	0 1 1 1 1 0 1 0 0 1 0 0 0 1 1	1437

Best Result: Profit Margin: 1442 and optimal solution is shown in table 5.34

Table 5.34: Optimal Solution of Test Case 4

Item	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Chromosome	0	1	1	0	1	1	1	0	0	1	0	0	1	0	1
Included?	n	y	y	n	y	y	y	n	n	y	n	n	y	n	y

5.1.5 Test Case 5:

Total number of items: 5

Total Money the buyer has (can be consider as weight of Knapsack): 26

Items Price (can be consider as item weights): 12, 7, 11, 8, 9

Items Margin (can be consider as profit):24, 13, 23, 15, 16

Population size: 4

Table 5.35: Output of Generation 1 of Test Case 5

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5	
1	0 0 1 1 0	38
2	0 0 1 0 1	39
3	0 0 1 1 0	38
3	0 0 1 1 0	38

Table 5.36: Output of Generation 2 of Test Case 5

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5	
1	0 0 1 1 0	38
2	0 1 1 1 0	51
3	0 0 1 1 0	38
4	0 0 1 0 1	39

Table 5.37: Output of Generation 3 of Test Case 5

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5	
1	0 0 1 1 0	38
2	0 1 1 1 0	51
3	0 0 1 0 1	39
4	0 0 1 0 1	39

Similarly the output upto 15^{th} generation is same as generation 5. No change in chromosomes after Generation 3.

Table 5.38: Output of Generation 15 of Test Case 5

S.NO	CHROMOSOMES	Margin
	Items 1 2 3 4 5	
1	0 0 1 1 0	38
2	0 1 1 1 0	51
3	0 0 1 0 1	39
4	0 0 1 0 1	39

Best Result: Profit margin: 51 and optimal solution is shown in table 5.39.

Table 5.39: Optimal Solution of Test Case 5

Item	1	2	3	4	5
Chromosome	0	1	1	1	0
Included?	no	yes	yes	yes	no

5.2 Comparison of Existing Algorithm versus Proposed Algorithm through Graph

The below graph shows the difference between performance of existing algorithm and proposed algorithm and shows how proposed algorithm proved better.

5.2.1 Test Case 1:

Both the algorithm i.e. existing and proposed had successive run for 4 times and the average result is shown in below graphs. The graph in fig 5.1 shows Maximum Margin as Profit variation along with 15 Generation.

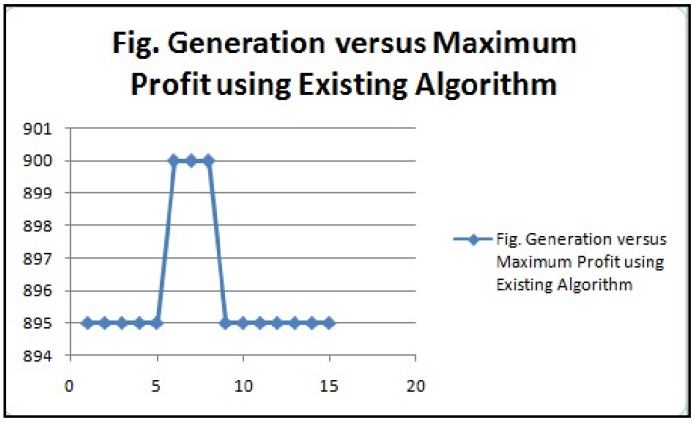


Fig 5.1: Graph of Existing Approach of Test case 1

It is clear that in the existing approach maximum margin can high in earlier generation and can fall down in later generation as shown above in figure the maximum profit is 900 in 5^{th} generation and become 895 in last generation.

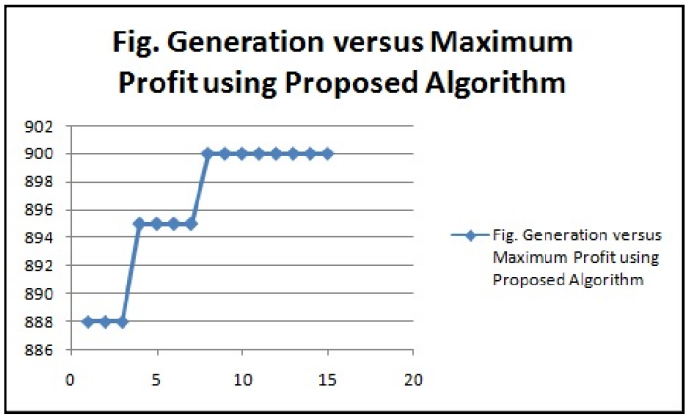


Fig 5.2: Graph of Proposed Approach of Test case 1

The proposed algorithm overcomes the drawback of existing algorithm by comparing intermediate and old population and with the help of graph shown in fig 5.2 it is proved that in the later generations maximum profit will raise instead of going down.

5.2.2 Test Case 2:

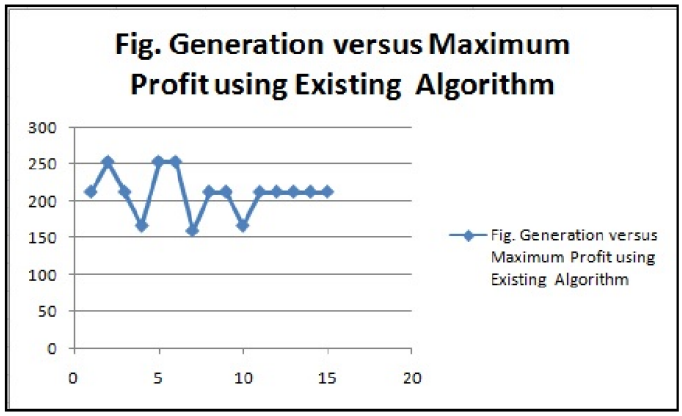


Fig 5.3: Graph of Existing Approach of Test case 2

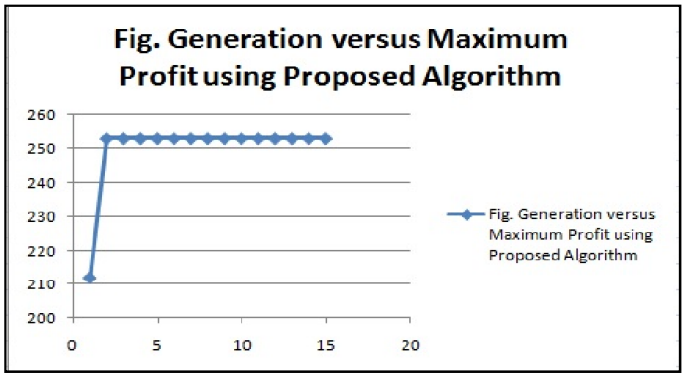


Fig 5.4: Graph of Proposed Approach of Test case 2

5.2.3 Test Case 3:

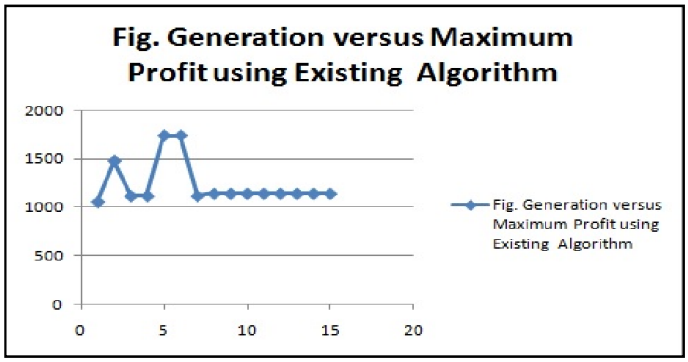


Fig 5.5: Graph of Existing Approach of Test case 3

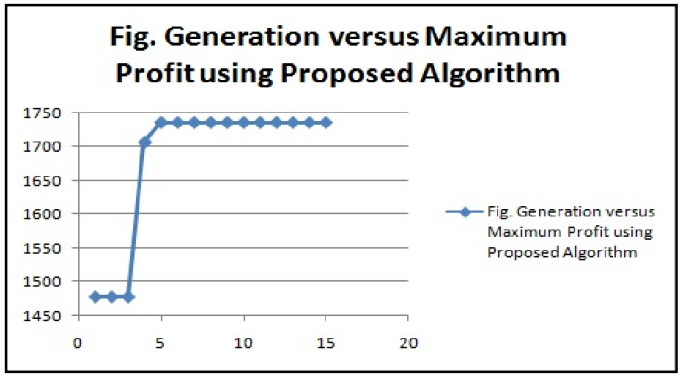


Fig 5.6: Graph of Proposed Approach of Test case 3

5.2.4 Test Case 4:

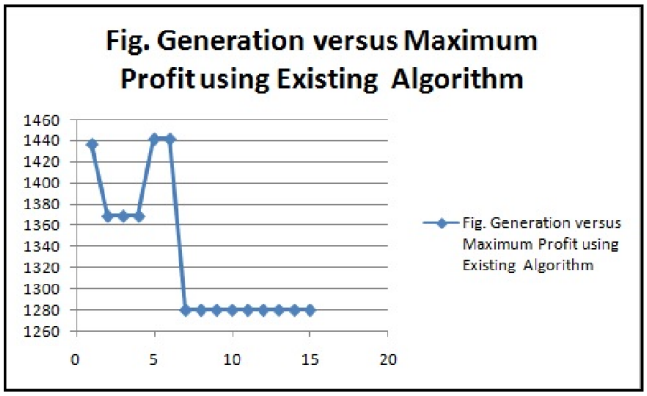


Fig 5.7: Graph of Existing Approach of Test case 4

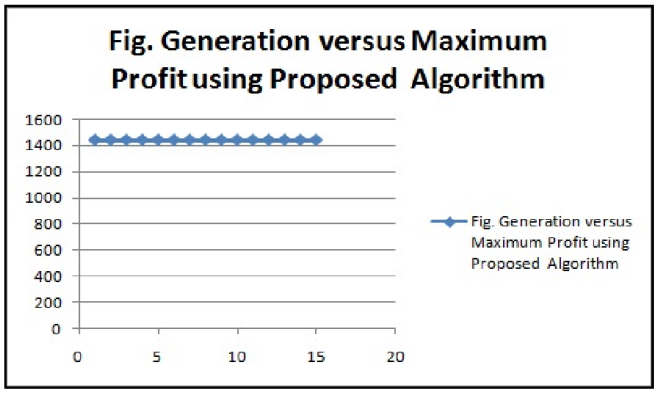


Fig 5.8: Graph of Proposed Approach of Test case 4

5.2.5 Test Case 5:

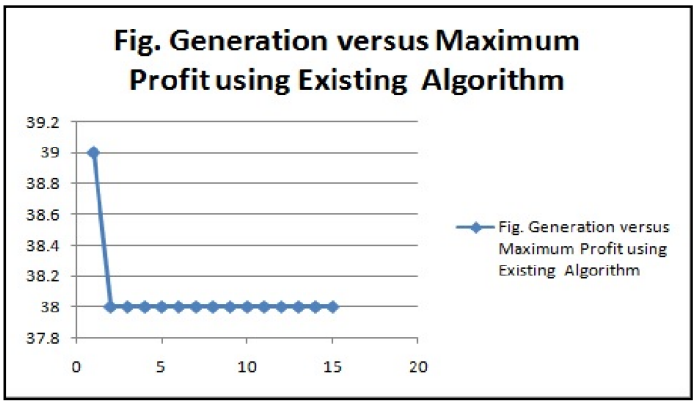


Fig 5.9: Graph of Existing Approach of Test case 5

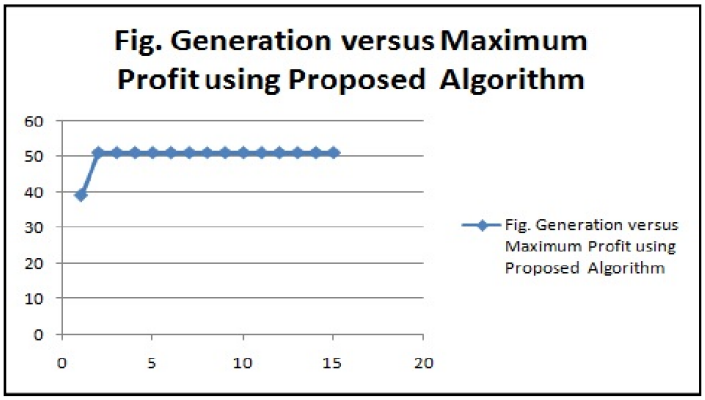


Fig 5.10: Graph of Proposed Approach of Test case 5

5.3 Snapshots:

The below snapshots was taken using test case 1.

The first screen that appears asks buyer to enter the total money that the buyer has and number of items.

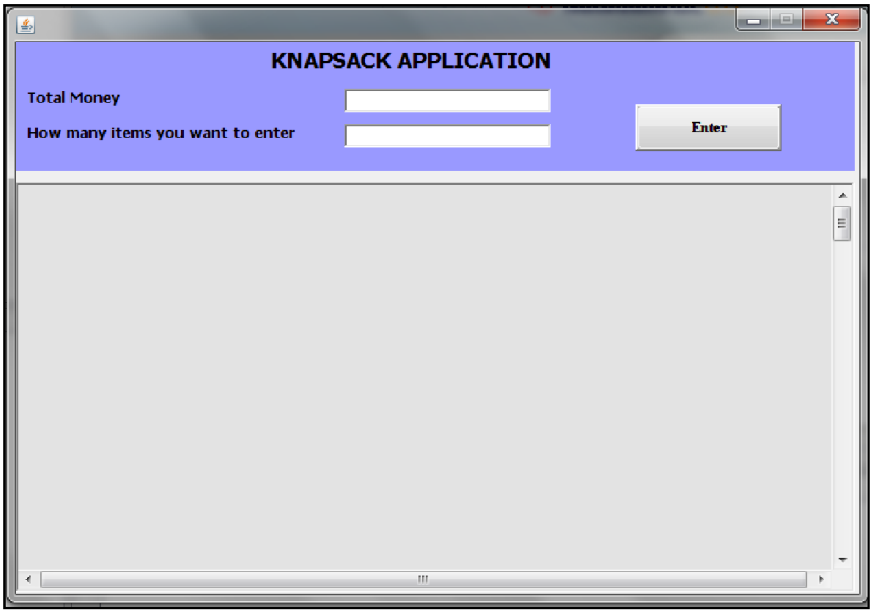


Fig 5.11: Enter Total Money and Number of Items

As soon as buyer enters the total money and number of items the next screen comes.

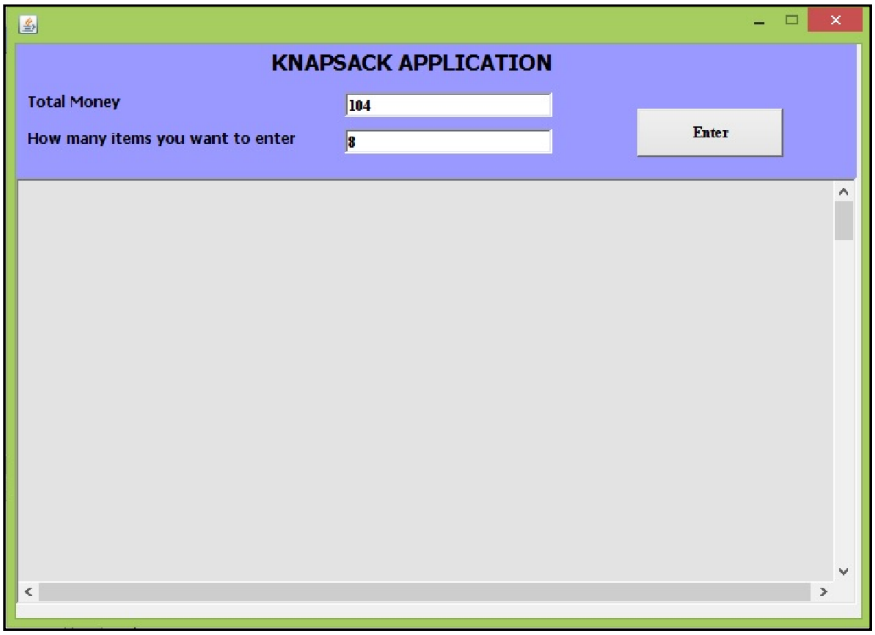


Fig 5.12: Entered Value of Total Money and Number of Items

After entering the values and after submitting the values the next screen comes

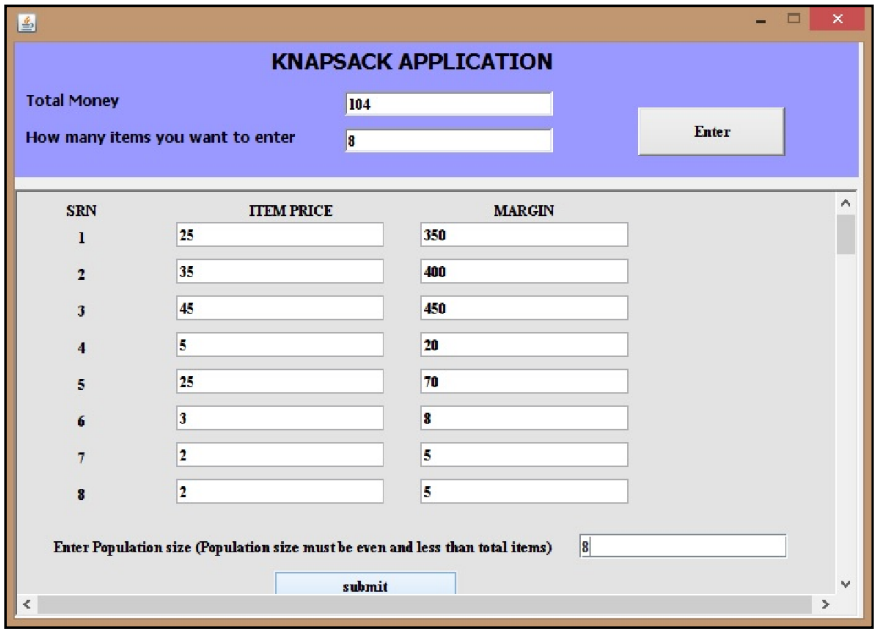


Fig 5.13: Enter the Details of Items

As soon as buyer enters the details of item then the output will come and result is displayed

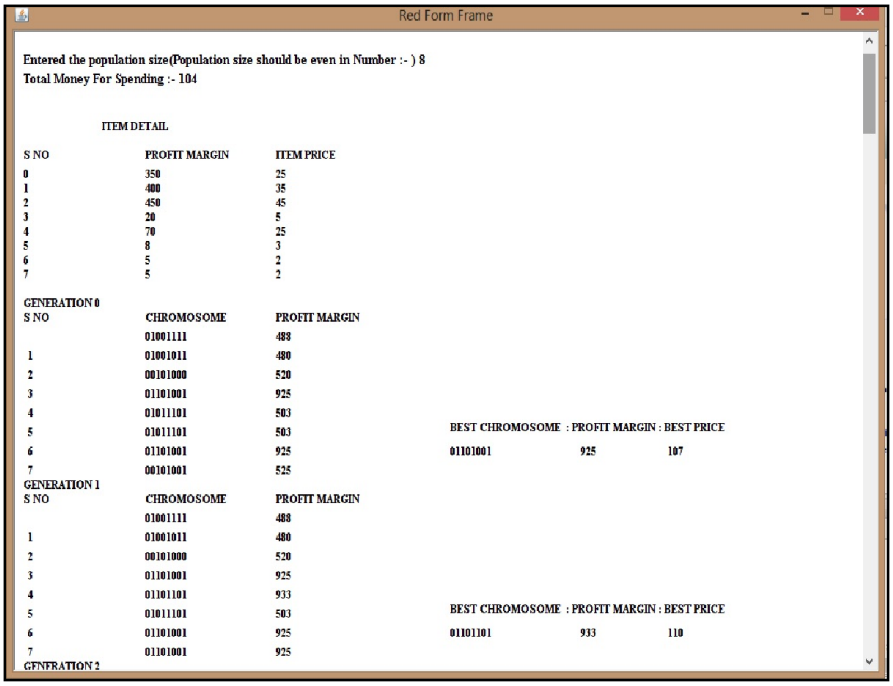


Fig 5.14: Output upto Generation 1

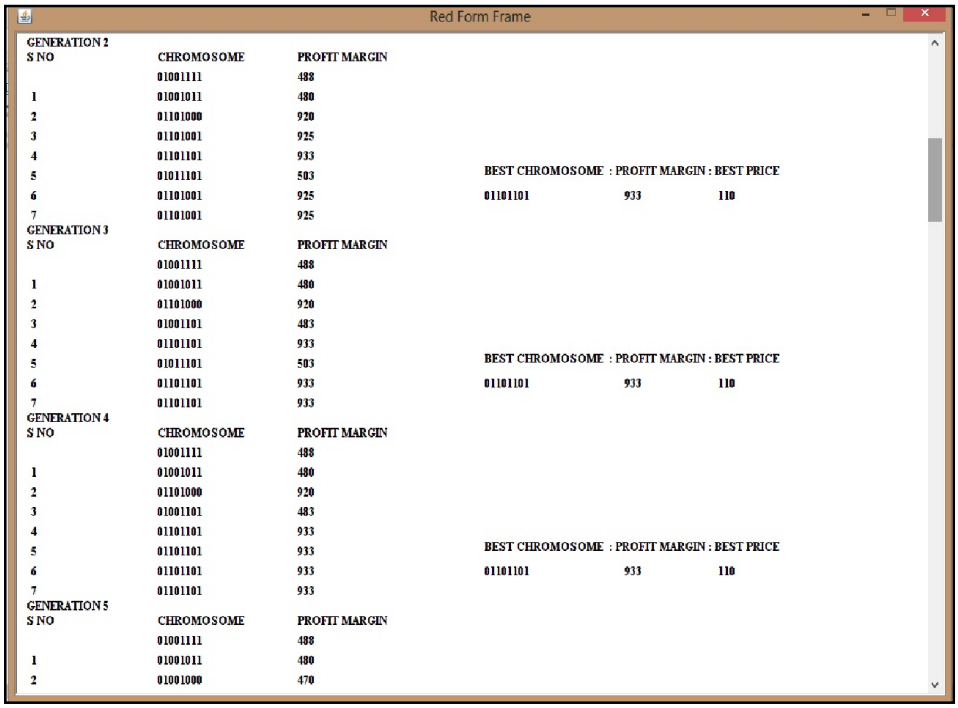


Fig 5.15: Output upto 6th Generation

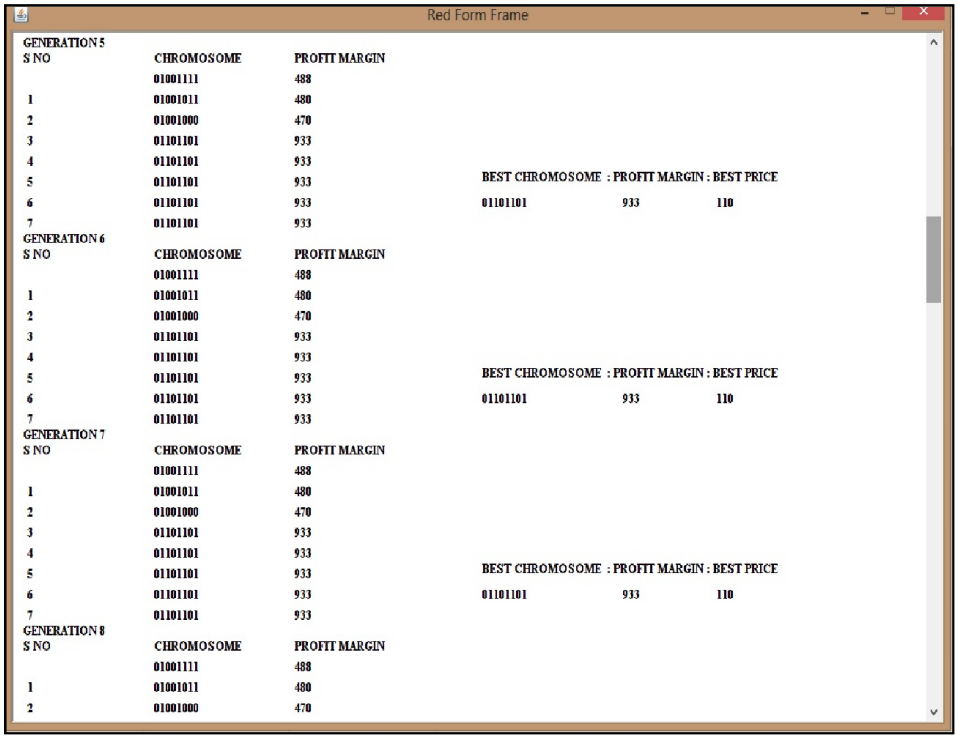


Fig 5.16: Output upto 7th Generation

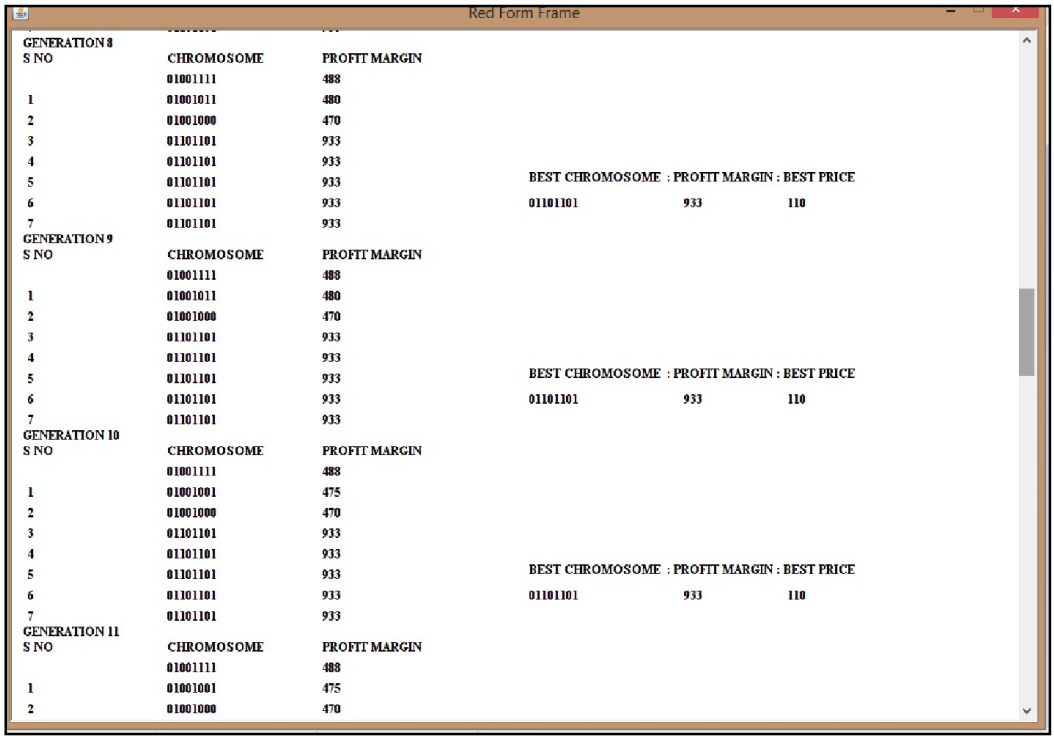


Fig 5.17: Output upto 10th Generation

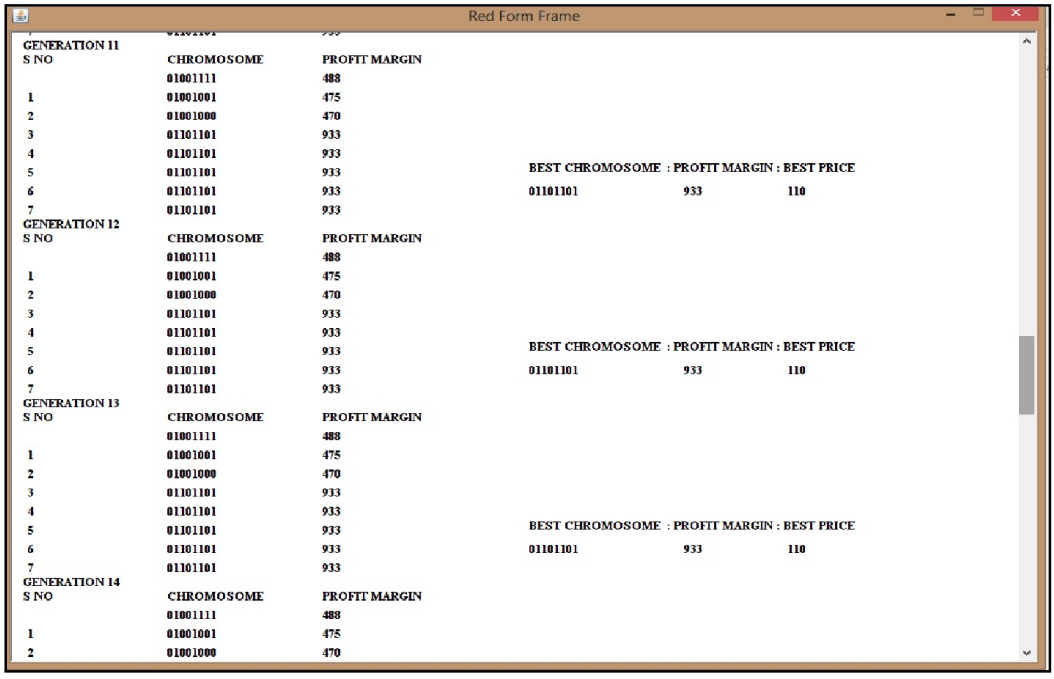


Fig 5.18: Output upto 13th Generation

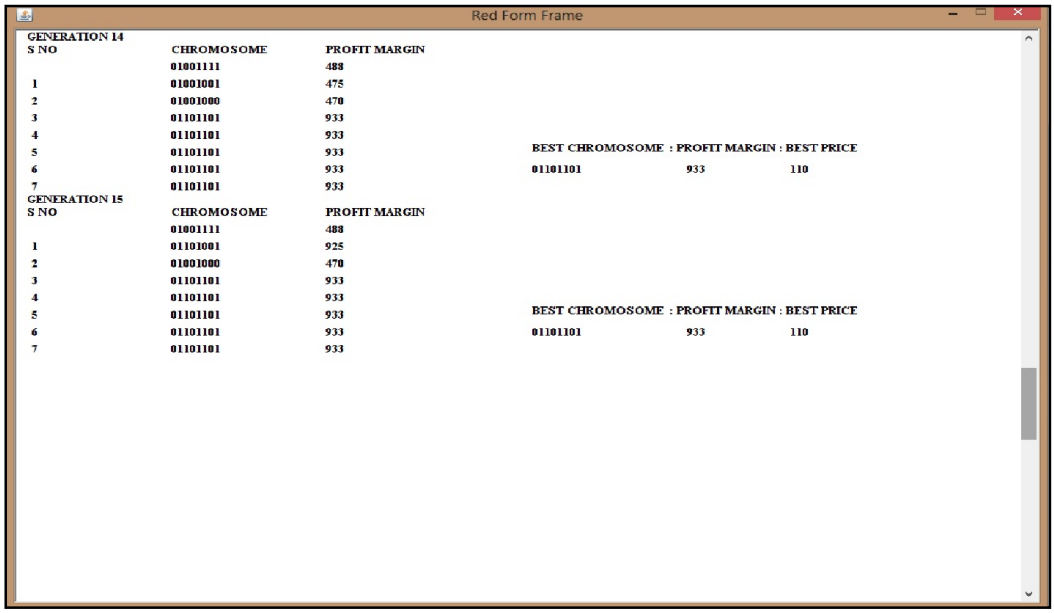


Fig 5.19: Output upto 15th Generation

CHAPTER 6: CONCLUSION AND FUTURE SCOPE

0/1 knapsack problem is an NP-complete problem. Many algorithms have been proposed for solving this problem such as Dynamic Programming and Greedy Strategy. Dynamic Programming considers all the possible solution for solving 0/1 knapsack problem and it gives optimal solution but it can't take polynomial time to solve the problem. Similarly Greedy Strategy takes optimal time to solve 0/1 knapsack problem but it can't guarantee to give optimal solution for this. Also this problem can be solved using Genetic Algorithms efficiently. The existing approach of genetic algorithm blindly replaces the older population with the new population created after crossover and mutation. The result of doing so is that the profit can be high in earlier generations and can be decreased in later generations. This shows the decrease in performance of solving 0/1 knapsack problem using genetic algorithms. To overcome this problem the proposed algorithm is given above. The proposed algorithm improved the performance of existing algorithm in the way that it can create an intermediate population. Now proposed algorithm compares the i^{th} chromosome of the intermediate population with the old population and after comparing the chromosomes whose fitness value from the both population is better than that will be a part of new generation i.e. The proposed algorithm check whether the newer generated chromosome have higher fitness or not than the older population. If yes then it will be a part of newer generation and if not then it will be rejected and chromosome will be copied from old generation and this all can be done in polynomial amount of time. In this way, better chromosomes i.e. higher fitness values in coming generations are generated. By doing so better results are obtained in the earlier generations and as the generations increase the fitness will get better from its earlier generation. Also variations in later generation will be smaller.

Although this algorithm improves the performance as compared to existing algorithms, but in future some learning techniques can be used for better results and also some other selection operator such as roulette wheel can be used to improve its performance.

REFERENCES

[1] E. Charniak and D. McDermott, " *Introduction to Artificial Intelligence*", Pearson, fourth edition, 2009

[2] A. Miller, "The structure, history and future of successful AI applications", *IEEE Trans. On Software Engineering*, Vol. 15, No. 8, pp. 12-14, Aug 1989.

[3] L. Xian, "Artificial Intelligence and Modren Sports Education Technology",*International Conference on Artificial Intelligence and Education(ICAIE)*, pp. 772-776, Oct 2010.

[4] Y. Dote and S. J. Ovaska, "*Industrial Applications of Soft Computing: A Review*", Proceedings of The IEEE, Vol. 89, pp. 1243-1265, September 2001.

[5] S.J. Ovaska, A. Kamiya, Y. Chen, "Fusion of Soft Computing and Hard Computing: Computational Structures and Characteristic Features", *IEEE Transactions On Systems, Man, And Cybernetics,*Vol. 36, pp. 439-448, May 2006.

[6] A.E. Eiben, J.E. Smith, "*Introduction to Evolutionary Computing*", Natural Computing Series, 2nd edition, 2007.

[7] M. Srinivas, L. M. Patnaik, "Genetic Algorithms: A Survey", *IEEE*, pp. 17-26 , 1994.

[8] M. A. Weiss, "*Data Structures & Algorithm Analysis in C++*," Addison-Wesley, Vol.424,_1999.

[9] M. Hristakeva and D. Shrestha, "Solving the 0-1 knapsack problem with genetic algorithms," *Midwest Instruction and Computing Symposium*, 2004.

[10] Non Deterministic Polynomial time Problem [online]. Available: http://dictionary.reference.com/browse/nondeterministic+polynomial+time

[11] L. Fortnow, " The Status of the P versus NP Problem", *Communication of the ACM,* vol. 52, pp. 78-86, sept 2009.

[12] S. Cook, "The Importance of the P versus NP Question", *Journal of the ACM*, vol. 50, No. 1, pp-27-29, Jan 2003.

[13] J.F. Zhao, T.L. Huang, F. Pang and Y. Liu, "Genetic algorithm based on greedy strategy in the 0-1 knapsack problem", *3rd International Conference on Genetic and Evolutionary Computing, IEEE*, pp. 105-107, 2009.

[14] D.S. Vianna, and J.E.C. Arroyo, "A GRASP algorithm for the multi-objective knapsack problem", *24th IEEE International Conference of the Chilean Computer Science Society*, pp. 69-75, 2004.

[15] R. Mahajan, S. Chopra, "Analysis of 0/1 Knapsack Problem using Deterministic And Probabilistic Techniques", *Second International Conference on Advanced Computing & Communication Technologies*, pp.150-155,2012.

[16] T. P. Patalia, G.R. Kulkarni, "Design of Genetic Algorithm for Knapsack Problem to perform Stock Portfolio Selection Using Financial Indicators", *International Conference on Computational Intelligence and Communication Systems,* pp. 289-292,2009.

[17] Greedy Srategy for knapsack problem [online]. Available: http://www.cse.ust.hk/~dekai/271/notes/L14/L14.pdf

[18] S. Kaystha, and S. Agarwal, "Greedy genetic algorithm to Bounded Knapsack Problem", *3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT)*, Vol. 6, pp. 301-305, 2010.

[19] B. C. Dean, M. X. Goemans, J. Vondrdk, "Approximating the stochastic Knapsack problem: the benefit of adaptivity", *Proceedings 45th Annual IEEE Symposium on Proc. Foundations of Computer Science*, pp. 208–217, 2004.

[20] R. P. Singh, "Solving 0-1 Knapsack Problem Using Genetic Algorithms", *IEEE*, pp 591-596, 2011.

[21] S. N. Mohanty and R. Satapathy, "An evolutionary multiobjective genetic algorithm to solve 0/1 Knapsack Problem", *IEEE International Conference on Computer Science and Information Technology*, pp. 397-399, 2009.

[22] H.H. Yang , S.W. Wang , H. T. Ko , J. C. Lin , "A Novel Approach for Crossover Based on Attribute Reduction – a Case of 0/1 Knapsack Problem", *Proceedings of the 2009 IEEE IEEM*, pp.1733-1737,2009.

[23] D. E. Goldberg, "*Genetic Algorithms in search , Optimization and Machine learning*", Pearson publication, 1989.

[24] B. H. Hasan, M. S. Mustafa*, "*Comparative Study of Mutation Operators on the Behavior of Genetic Algorithms Applied to Non-deterministic Polynomial (NP) Problems", *2011 Second International Conference on Intelligent Systems, Modelling and Simulation*, pp. 7-12, 2011.

[25] M. Mitchell, "*An Introduction to Genetic Algorithms*", MIT Press, First Edition, 1998.

[26] D. E. Goldberg, J. H. Holland, "*Genetic Algorithms and Machine Learning"* Kluwer Academic Publishers-Plenum, vol. 3, Oct 1988.

[27] M. Srinivas, L. M. Patnai , "Adaptive probabilities of crossover and mutation in genetic algorithms", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 24, No. 4, pp. 656-666, 1994.

[28] G. Rudolph, "Self –adaptive mutations may lead to premature convergence", *IEEE Transactions on Evolutionary Computation*, Vol. 5, No. 4, pp. 410-414, Aug 2001.

[29] L. Meiyi, C. Zixing and S. Gurong, "An adaptive genetic algorithm with diversity –guided mutation and its global convergence property", *Journal CSUT*, Vol.11, No.3, 2004.

[30] M. Yanqin, "The Modified Hybrid Adaptive Genetic Algorithm For 0-1 Knapsack Problem", *IEEE 24th Chinese Control and Decision Conference (CCDC)*, pp. 326-329, 2012.

[31] M. Guevara-Souza, "An Approach for the Knapsack Problem Using Genetic Algorithms with Learning Capabilities", *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC 2009)* , pp.330-335,2009.

[32] H. Schildt, "*Java 2: The Complete Reference*", Fifth edition, McGraw-Hill, 2002.

[33] K. Sierra, B. Bates, "*Head First Java*", Second edition, O'Reilly Media , February 2005.

[34] H. Bock, "*The Definitive Guide to NetBeans Platform 7*", Apress, 2011.

[35] Test Case [online].
Available:http://people.sc.fsu.edu/~%20jburkardt/datasets/knapsack_01/knapsack_01.html

LIST OF PUBLICATIONS

[1] C. Sachdeva, S. Goel, "An Improved Approach for Solving 0/1 Knapsack Problem In Polynomial Time Using Genetic Algorithms", In Proceedings *IEEE International Conference on Recent Advances and Innovations in Engineering (ICRAIE-2014)*, Jaipur India, May 09-11, pp. 41 , 2014(is to be published by IEEE Explore).

[2] C. Sachdeva, S. Goel, "Adaptive Genetic Algorithm Versus Modified Hybrid Adaptive Genetic Algorithm for Solving 0/1 Knapsack Problem", *communicated (March 2014) in Information Resources Management Journal* (*(IGI Global)*), USA, 2014.