Literature Review On Implementing Binary Knapsack problem

Ms. Niyati Raj, Prof. Jahnavi Vitthalpura

PG student Department of Information Technology,
L.D. College of Engineering, Ahmedabad, India
Assistant Professor at Department of Information Technology,
L.D. College of Engineering, Ahmedabad, India

Abstract

Knapsack problem is a traditional combinatorial optimization problem which aims to maximize the payload without exceeding the capacity of the bag. When one of the problem variables which are "the capacity of the bag" or "the types/numbers of materials" is increased, the complexity of the problem size increases significantly. Because of the complexity of this problem, it has been become an area of interest for researchers and is intended to be solved with different approaches. Among these different solution approaches without check-ing in all search space, Evolutionary algorithms reveals approaches which are acceptable compliance solutions with producing an acceptable period of time.

Keywords: *0/1 Knapsack Problem, Genetic Algorithm, Rough Set Theory, Ant Weight Lifting Algorithm*

Introduction

Let i be the highest-numbered item in an optimal solution S for W pounds. Then S‘ = S - i is an optimal solution for W - wi pounds and the value to the solution S is Vi plus the value of the subproblem. We can express this fact in the following formula: define c[i, w] to be the solution for items 1,2, . . . , i and maximum weight w. Then

c[i,w] = 0	c[i-1, w] max [vi + c[i-1, w-wi] c[i-1, w]	if i = 0 or w = 0 if wi ¸ 0 if i>0 and w ¸ wi

This says that the value of the solution to i items either include i^{th} item, in which case it is vi plus a subproblem solution for (i - 1) items and the weight excluding wi, or does not include i^{th} item, in which case it is a subproblem’s solution for (i - 1) items and the same weight. That is, if the thief picks item i, thief takes vi value, and thief can choose from items w - wi, and get c[i - 1, w - wi] additional value. On other hand, if thief decides not to take item i, thief can choose from item 1,2, . . . , i- 1 upto the weight limit w, and get c[i - 1,w] value. The better of these two choices should be made. Although the 0-1 knapsack problem, the above formula for c is similar to LCS formula: boundary values are 0, and other values are computed from the input and "earlier" values of c. So the 0-1 knapsack algorithm is like the LCS-length algorithm given in CLR for finding a longest common subsequence of two sequences. The algorithm takes as input the maximum weight W, the number of items n, and the two sequences v = <v1, v2, . . . , vn> and w = <w1, w2, . . . , wn>. It stores the c[i, j]values in the table, that is, a two dimensional array, c[0 . . n, 0 . . w] whose entries are computed in a row-major order. That is, the first row of c is filled in from left to right, then the second row, and so on. At the end of the computation, c[n, w] contains the maximum value that can be picked into the knapsack.

Dynamic-0-1-knapsack (v, w, n, W)

```
FOR w = 0 TO W
DO c[0, w] = 0
    FOR i=1 to n
DO c[i, 0] = 0
    FOR w=1 TO W
DO If wi · w
THEN IF vi + c[i-1, w-wi]
THEN c[i, w] = vi + c[i-1, w-wi]
ELSE c[i, w] = c[i-1, w]
ELSE
c[i, w] = c[i-1, w]
```

The set of items to take can be deduced from the table, starting at c[n. w] and tracing backwards where the optimal values came from. If c[i, w] = c[i-1, w] item i is not part of the solution, and we are continue tracing with c[i-1, w]. Otherwise item i is part of the solution, and we continue tracing with c[i-1, w-W].

Analysis

This dynamic-0-1-kanpsack algorithm takes μ(nw) times, broken up as follows: μ(nw) times to fill the c-table, which has (n +1).(w +1) entries, each requiring μ(1) time to compute. Ø(n) time to trace the solution, because the tracing process starts in row n of the table and moves up 1 row at each step. Following are some of the approaches to solve knapsack problem:

1 Solve Zero-One Knapsack Problem by Greedy Genetic Algorithm

Genetic Algorithm:

A Genetic Algorithm (GA) is an algorithm that mimics the process of natural selection. In a genetic al-gorithm, a population of candidate solutions (called individuals, creatures, or phenotypes) to a problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or geno-type) which can be mutated and altered; traditionally, solutions are represented in binary as strings of Os and 1 s, but other encodings are also possible. The evolution usually starts from a population of randomly gen-erated individuals, and is an iterative process, with the population in each iteration called a generation.The fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. Each iteration of the algorithm is called generation.

In this article, for the genetic algorithm, the group number M, the terminating evolution generation T, crossover probability Pc and mutation probability Pm are supposed to be M =100, T =100, Pc =0.5 and Pm =0.05.

D. Experiments and Analyses

Experiment 1 [10][11]:

n=10, C=269, W=95, 4, 60, 32, 23, 72, 80, 62, 65, 46, V=55, 10, 47, 5,4, 50, 8, 61, 85, 87

Experiment 2

n=20, C =878,

W=92, 4, 43, 83, 84, 68, 92, 82, 6, 44, 32, 18, 56, 83, 25, 96, 70, 48, 14, 58,

V=44, 46, 90, 72, 91, 40, 75, 35, 8, 54, 78, 40, 77, 15, 61, 17, 75, 29, 75, 63

Experiment 3

n=50, C= 1000,

W=220, 208, 198, 192, 180, 180, 165, 162, 160, 158,155, 130, 125, 122, 120, l18, l15, 110, 105, 101, 100, 100, 98, 96, 95, 90, 88, 82, 80, 77, 75, 73, 72, 70, 69, 66, 65, 63, 60, 58, 56, 50, 30, 20, 15, 10, 8, 5, 3, 1,
V=80, 82, 85, 70, 72, 70, 66, 50, 55, 25, 50, 55, 40,48, 50, 32, 22, 60, 30, 32, 40, 38, 35, 32, 25, 28, 30, 22, 25, 30, 45, 30, 60, 50, 20, 65, 20, 25, 30, 10, 20, 25, 15, l0, l0, 10, 4, 4, 2, 1.

Experiment 4 [12]

n=100, C =4995,

W=108, 98, 95, 107, 98, 100, 96, 105, 93, 112, 95,105, 91, 96, 100, 103, 91, 96, 105, 90, 101, 110, 108, 95,99, 96, 108, 101,102, 100, 111, 88, 99, 112, 101, 105, 94, 113, 87, 101, 108, 96, 91, 89, 102, 99, 98, 93, 98, 99, 106, 112, 90, 100, 92, 94,98, 97, 99, 95, 112, 108, 100, 98, 117, 98, 100, 98, 99, 113, 94, 111, 102, 99, 97, 87, 97, 103, 97, 89, 96, 94, 93, 104, 92,109, 97, 109, 100, 88, 92, 108, 97, 106, 97, 97, 99, 94, 102, 95

V=117, 113, 113, 113, 112, 112, 112, 112, 112, 111,110, 110, 109, 109, 108, 108, 108, 108, 108, 108, 108, 107, 106, 106, 105,105, 105, 105, 104, 103, 102, 102, 102, 101, 101, 101, 101, 100, 100, 100, 100, 100, 100, 99, 99, 99, 99, 99, 99, 99, 99, 98,98, 98, 98, 98, 98, 98, 98, 97, 97, 97, 97, 97, 97, 97, 97, 96, 96, 96, 96, 96, 96, 95, 95, 95, 95, 95, 94, 94, 94, 94, 94, 93, 93,93, 92, 92, 92, 91, 91, 91, 90, 90, 89, 89, 88, 88, 87, 87.

Greedy algorithm (GA), standard GA (SGA) and GGA did ten times according to the above four experiments with the results shown in table 1.

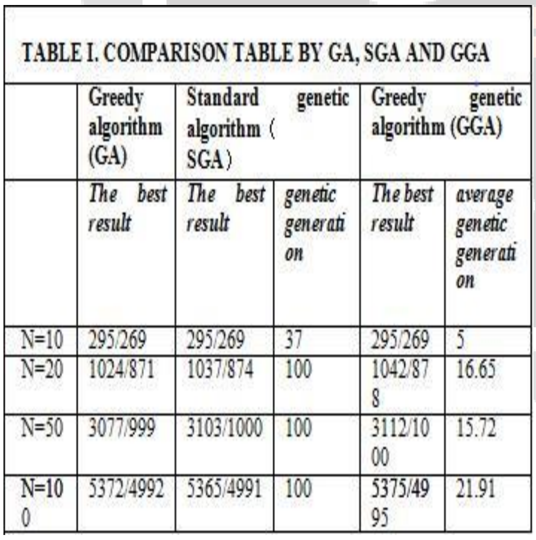
TABLE I. COMPARISON TABLE BY GA, SGA AND GGA

	Greedy algorithm (GA)	Standard genetic algorithm (SGA)		Greedy genetic algorithm (GGA)	
	The best result	*The best result*	*genetic generation*	*The best result*	*average genetic generation*
N=10	295/269	295/269	37	295/269	5
N=20	1024/871	1037/874	100	1042/878	16.65
N=50	3077/999	3103/1000	100	3112/1000	15.72
N=100	5372/4992	5365/4991	100	5375/4995	21.91

From Table 1, we can analyse as follows: (1) To the knapsack problem with 10 items, the greedy algorithm, standard genetic algorithm, and greedy genetic algorithm all got the best result. GGA not only has high convergence, but also has rapid convergence speed. GGA can get the optimal solution when the average genetic generation is 5.

(2) To the knapsack problem with 20 items, the SGA got good result. However, GGA got better result. Not only is the convergence of GGA quite high, but also the convergence speed is quite rapid. GGA can get the optimal solution when the average genetic generation is 16.65.
(3) To the knapsack problem with 50 items, both of the SGA and GGA succeed. Nevertheless, GGA is superior to the SGA in precision and convergence speed. GGA can get the optimal solution when the average genetic generation is 15.72.
(4) To the knapsack problem with 100 items, GGA is Superior to SGA in precision (The worst precision of GGA is higher than the average precision of SGA. Meanwhile, the SGA precision is worse than that of GA sometimes.) Moreover, the convergence speed of GGA is faster than that of SGA (GGA can get the optimal solution when the average genetic generation is 21.91. However, that of SGA is 100).
The above examples prove that the computing result of GGA is smartly superior to GA and SGA. The reason is that the greedy transform makes GGA have higher probability to get excellent coding chromosome. Thereby, using GGA to solve the zero-one knapsack problem will achieve better effect.

2 Solving the 0-1 Knapsack Problem Using Genetic Algorithm and Rough Set Theory

In the ordinary set theory, crisp sets are used. A set is then defined uniquely by its elements, i.e., to define a set we have to point out its elements. The membership function, describing the belongingness of elements of the universe to the set, can attain one of the two values, 0 or 1 [41. 1 indicates that the element belongs to the set and 0 indicates that it does not. A fuzzy set is defmed by the membership function which can attain values from the closed interval [0,1] , allowing partial membership of the elements in the set. A rough set is a formal approximation of a crisp set (i.e., conventional set) in terms of a pair of sets which give the lower and the upper approximation of the original set[41. In the standard version of rough set theory, the lower and upper approximation sets are crisp sets, but in other variations, the approximating sets may be fuzzy sets.

• The lower approximation, or positive region, is the union of all equivalence classes in which are contained by (i.e., are subsets of) the target set.
• The upper approximation is the union of all equivalence classes in which have non-empty intersection with the target set
• The boundary region consists of those objects that can neither be ruled in nor ruled out as members of the target set.
Thus, a rough set is composed of two crisp sets, one representing a lower boundary of the target set, and the other representing an upper boundary of the target set.

3 Solving 0/1 Knapsack Problem using Ant Weight Lifting Algorithm

In here, Knapsack problem is solved using the ant weight lifting algorithm (AWL). Suppose, the total no. of items is equal to N. Each item (i=1 to N) is characterized by profit i and weight i. Knapsack problem deals with the selection of k items among N items. The selection maximizes the total profit, such that, it does not exceed the knapsack capacity, which is formally known as bounded knapsack problem. Here k no. of ants are chosen with capacity of 50, which indicates the knapsack capacity. Each ant can collect upto 50 food grains and go back to the central food repository (CFR). Each set of ants collects food without violating the problem definition itself. Maximum iteration is denoted by $\max_i tr$.

Step 1: while (itr=0 to $\max_i tr$)

Step 2: k number of ants are randomly placed.

Step 3: Each item is randomly collected if it is not already collected by another ant.

Step 4: Each ant collects food within its capacity having maximum profit (Calorific value).

Step 5: The best ant is detected which collects food with maximum calorific value.

Step 6: If one ant finds the best solution, then the global best solution is updated.
Step 7: Step 2 to step 5 is repeat upto $\max_i tr$

In here, Ant Algorithm has been applied to solve 0-1 Knapsack Problem and to find the maximum profit without exceeding the knapsack size. In this study Ant algorithm has been applied on three different data sets. Each dataset contains different number of items with different weights and profits. Dataset 1, Dataset 2 and Dataset 3 contain 100, 200 and 500 items respectively with different weight and profit. Genetic algorithm has been also applied to compare with AWL. From Fig. 2, Fig. 3 and Fig. 4 it is clear that in every dataset, AWL gives better performance than the genetic algorithm. In respect to time it also gives better result than GA. In GA, three GA operator selection, crossover and mutation take time to find the optimum solution (here maximization of profit, keeping the total weight within the knapsack capacity) in the population. But the proposed approach considers the finite number of ants, each ant collects knapsack upto its maximum weight lifting capacity. As soon as an ant reaches its maximum limit, its total profit can be measured. It may not be an optimized solution but still it is a valid solution. So, it works with each valid solution and the best solution is found out.

Conclusion

The Genetic Algorithm provides a way to solve the knapsack problem in linear time complexity, as opposed to dynamic programming which has an exponential time complexity. Knowing which attributes are important reduces the search space. Also, important genes ensure the effective information will not be lost or destroyed in the evolution. The hybrid approach produced a better solution than the genetic algorithm. The greedy ge-netic algorithm quickens the searching speed, increases the precision and overcomes the disadvantage of the traditional algorithm effectively which is easily running into local optimization.

References

1.1 Veenu Yadav And Ms Shikha Singh,"A review paper on solving 0-1 Knapsack Problem with Genetic Algo-rithm"

1.2 Tribikram Pradhan And Akash Israni,"Solving the 0-1 Knapsack Problem using Genetic Algorithm and Rough Set Theory"

1.3 Sourav Samanta And Sayan Chakraborty,"Solving 0/1 Knapsack problem using Ant Weight Lifting Prob-lem"