

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/301409726>

Solving the 0–1 Knapsack problem using Genetic Algorithm and Rough Set Theory

Conference Paper · May 2014

DOI: 10.1109/ICACCT.2014.7019272

CITATIONS

11

READS

8,138

3 authors, including:



Tribikram Pradhan

Indian Institute of Technology BHU

40 PUBLICATIONS 445 CITATIONS

SEE PROFILE

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

Tribikram Pradhan¹, Akash Israni², Manish Sharma³

^{1,2,3}Department of Information and Communication Technology (ICT), Manipal University, Manipal – 576104, Karnataka, India

¹tribikram.pradhan@manipal.edu, ²akashisrani12@gmail.com, ³manishsharma7207@gmail.com

Abstract—This paper describes a hybrid algorithm to solve the 0-1 Knapsack Problem using the Genetic Algorithm combined with Rough Set Theory. The Knapsack problem is a combinatorial optimization problem where one has to maximize the benefit of objects in a knapsack without exceeding its capacity. There are other ways to solve this problem, namely Dynamic Programming and Greedy Method, but they are not very efficient. The complexity of Dynamic approach is of the order of $O(n^3)$ whereas the Greedy Method doesn't always converge to an optimum solution^[2]. The Genetic Algorithm provides a way to solve the knapsack problem in linear time complexity^[2]. The attribute reduction technique which incorporates Rough Set Theory finds the important genes, hence reducing the search space and ensures that the effective information will not be lost. The inclusion of Rough Set Theory in the Genetic Algorithm is able to improve its searching efficiency and quality.

Keywords—Genetic Algorithm (GA), Rough Set Theory, Attribute reduction Techniques, Knapsack Problem, 0-1 Knapsack Problem.

I. INTRODUCTION

A. The Knapsack Problem

The Knapsack problem is a combinatorial optimization problem where one has to maximize the benefit of objects in a knapsack without exceeding its capacity. Given a set of items we have to find optimal packing of a knapsack. Each item is characterized by its weight and profit value and the knapsack is characterized by its capacity. Optimal packing is the one in which the total weight is less than or equal to the capacity and in which value is maximal among other feasible packings.

B. Genetic Algorithm

A Genetic Algorithm (GA) is an algorithm that mimics the process of natural selection. In a genetic algorithm, 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 genotype) which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible.

The evolution usually starts from a population of randomly generated individuals, and is an iterative process, with the population in each iteration called a generation. In each generation, the fitness of every individual in the population is

evaluated; 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.

C. 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^[4]. 1 indicates that the element belongs to the set and 0 indicates that it does not.

A *fuzzy set* is defined 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^[4]. 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.

II. EXISTING SYSTEM

The 0-1 knapsack problem can be solved by the genetic algorithm.

A. An Example of a 0-1 Knapsack Problem

Suppose we have a knapsack that has a capacity of 13 cubic inches and several items of different weights and different profit values. We want to include in the knapsack only those items that will have the greatest total profit within the constraint of the knapsack's capacity. There are three potential items (labeled 'A', 'B', 'C'). Their weight and profit values are as follows:

Item (X)	A	B	C
Profit (P)	4	3	5
Wight (W)	6	7	8

We seek to maximize the total benefit:

$$\sum_{i=1}^3 P_i X_i = 4X_1 + 3X_2 + 5X_3$$

Subject to the constraints:

$$\sum_{i=1}^3 W_i X_i = 6X_1 + 7X_2 + 8X_3 \leq 13$$

And

$X_i \in \{0,1\}$; for $i = 1, 2, \dots, n$.

For this problem there are $2^3 = 8$ possible ways to select items:

A	B	C	Total Weight	Total Profit
0	0	0	0	0
0	0	1	8	5
0	1	0	7	3
0	1	1	15	-
1	0	0	6	4
1	0	1	14	-
1	1	0	13	7
1	1	1	21	-

In order to find the best solution we have to identify a subset that meets the constraint and has the maximum total benefit. In our case, only rows given in italics satisfy the constraint. Hence, the optimal benefit for the given constraint can only be obtained with one quantity of A, one quantity of B, and zero quantity of C, and it is 7. Thus, the row in bold is the solution.

B. Basic Elements of the Genetic Algorithm

Most GA methods are based on the following elements: populations of chromosomes, selection according to fitness, crossover to produce new offspring, and random mutation of new offspring.

Chromosomes: The chromosomes in GAs represent the space of candidate solutions. There are many types of chromosome encodings. For the Knapsack problem, we use binary encoding, where every chromosome is a string of bits, 0 or 1.

Encoding of chromosomes: A chromosome can be represented in an array having size equal to the number of the items (in our example, the size is 3). Each element from this array denotes whether an item is included in the knapsack ('1') or not ('0')^[1]. For example, the following chromosome:

A	B	C
1	0	1

indicates that the 1st and the 3rd item (A and C) are included in the knapsack.

Fitness Function: GAs require a fitness function which allocates a score to each chromosome in the current population. Thus, it can calculate how well the solutions are coded and how well they solve the problem. For solving the 0-1 knapsack problem using genetic algorithm, the fitness function of a chromosome is the total profit of the chromosome. For example, consider the chromosome:

A	B	C
1	1	0

The profit values of A, B and C are 4, 3 and 5 respectively. Since only A and B are included in the knapsack, the fitness of the above chromosome will be the sum of the profit of A and B. Hence, we have:

Fitness of the chromosome = $4 + 3 = 7$.

Selection: The selection process is based on fitness.

Chromosomes that are evaluated with higher fitness values will most likely be selected to reproduce, whereas, those with low values will be discarded. The fittest chromosomes may be selected several times, however, the number of chromosomes selected to reproduce is equal to the population size, therefore, keeping the size constant for every generation. This phase has an element of randomness just like the survival of organisms in nature. There are different types of selection methods. In our implementation, we have used roulette-wheel selection.

Crossover: Crossover is analogous to biological reproduction. It is the process of combining the bits of one chromosome with those of another. This is to create an offspring for the next generation that inherits traits of both parents. Crossover randomly chooses a locus and exchanges the subsequences before and after that locus between two chromosomes to create two offspring. For example, consider the following parent chromosomes and a crossover point at position 3:

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

Mutation: It is analogous to biological mutation. It changes the new offspring by flipping bits from 1 to 0 or from 0 to 1. It is done to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next. For example, consider the following chromosome with mutation point at position 2:

Original chromosome: 1 0 0 0 1 1 1

Mutated chromosome: 1 1 0 0 1 1 1

The 0 at position 2 flips to 1 after mutation.

C. Outline of basic Genetic Algorithms

- 1) Start: Randomly generate a population of N chromosomes.
- 2) Fitness: Calculate the fitness of all chromosomes.
- 3) Create a new population:
 - a) Selection: According to the selection method select 2 chromosomes from the population.
 - b) Crossover: Perform crossover on the 2 chromosomes selected.
 - c) Mutation: Perform mutation on the chromosomes obtained.
- 4) Replace: Replace the current population with the new population.
- 5) Test: Test whether the end condition is satisfied. If so, stop. If not, return the best solution in current population and go to Step 2.

Each iteration of this process is called a generation.

III. PROPOSED SYSTEM

The Genetic Algorithm consists of 7 different steps:

1. Generation of initial population
2. Fitness function
3. Selection
4. Crossover
5. Mutation
6. Replacement
7. Termination

In our method, we have introduced an eighth step called “Attribute reduction technique”. The attribute reduction technique incorporates rough set theory to calculate the importance of different attributes and hence, produce a more optimal solution. After each iteration of the genetic algorithm, this step will be carried out. The final population resulting from one iteration is used as the initial population for the next iteration.

IV. METHODOLOGY

Figure 1 depicts the sequence of steps carried out in the algorithm. The attribute reduction technique (which uses

rough set theory) is carried out after every iteration of the genetic algorithm.

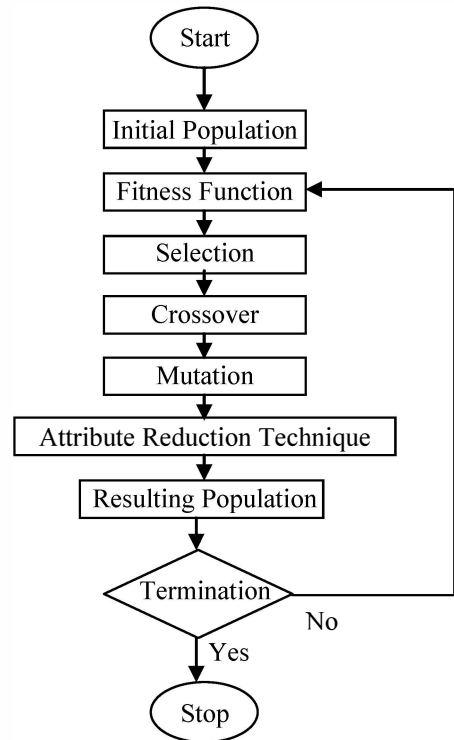


FIGURE 1. FLOWCHART OF THE ALGORITHM

V. EXPERIMENTATION

For example, consider the following knapsack problem:

- Take $n=4$, where n is the number of items.
- $(P_1, P_2, P_3, P_4) = (8, 1, 4, 2)$ where P_1, P_2, P_3, P_4 are profit values of respective items.
- $(W_1, W_2, W_3, W_4) = (2, 4, 5, 1)$ where W_1, W_2, W_3, W_4 are weights of respective items.
- $C=7$, where C is the capacity of the knapsack.

Table 1 shows the 12 possible ways to select the 4 items C_1, C_2, C_3 , and C_4 :

TABLE 1. SAMPLE SOLUTIONS

S. no.	C_1	C_2	C_3	C_4
1	0	0	0	1
2	1	0	0	1
3	0	0	1	0
4	1	0	1	0
5	0	1	0	1
6	1	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	1	1	1
10	1	0	1	1
11	0	0	1	1
12	1	1	0	0

The 6th, 7th, 9th and 10th rows are unfit chromosomes as their total weight exceeds the capacity of the knapsack, so they are not included in Table 2.

TABLE 2. CROSSOVER AND MUTATION

	Crossover Index	Crossover	Mutation (C)	Fitness	Group (D)
0001	2	0001	0101	3	2
1001	2	1001	1000	8	2
1010	3	1011	1001	10	2
0101	3	0100	0110	5	1
1000	1	1011	1001	10	2
0011	1	0000	0100	1	1
1100	2	1110	1010	12	2
0010	2	0000	0100	1	1

Crossover and mutation are carried out. The fit items are put in group 1 and unfit ones are put in group 2. We analyze the relationship between the gene position and the fitness function by using the concepts of positive region and attribute importance. Then the important genes (attributes) are found.

$$U / \text{ind}(C) = \{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}\}$$

$$U / \text{ind}(D) = \{\{1,2,3,5,7\}, \{4,6,8\}\}$$

$$\square_c(D) = \text{pos}_c(D) = 1$$

Therefore Table 2 is consistent.

In Table 3, we remove C_1 , and hence are left with only C_2, C_3 and C_4 .

TABLE 3. REMOVING C_1

S.no	C - C_1	Crossover	Mutation (C)	Fitness	Group (D)
1	001	001	101	3	2
2	001	001	011	6	1
3	010	010	011	6	1
4	010	010	110	5	2
5	101	101	111	7	2
6	110	110	010	4	1
7	111	110	100	1	2
8	000	001	000	0	2
9	111	011	111	7	2
10	011	111	101	3	2
11	011	010	011	6	1
12	100	101	001	2	2
13	101	001	101	3	2
14	000	100	110	5	2
15	001	000	001	2	2
16	110	111	011	6	1
17	001	101	111	7	2
18	100	000	001	2	2
19	010	010	110	5	2
20	100	100	110	5	2

$$\text{Average fitness} = 85/20 = 4.25 \approx 4$$

$$U/\text{ind}(C-C_1) = \{\{1,2,15,17\}, \{3,4,19\}, \{5,13\}, \{6,16\}, \{7,9\}, \{8,14\}, \{18,20\}\}$$

$$\text{Pos}_{\{C-C_1\}}(D) = \{2,3,6,11,16\} \text{ (5 items in Group 1)}$$

Therefore, the importance of C_1 :

$$\omega_{c1} = 1 - (5/20) = 15/20 = 3/4 = 0.75$$

TABLE 4. REMOVING C_2

S.no	C - C_2	Crossover	Mutation (C)	Fitness	Group (D)
1	001	101	001	2	2
2	101	001	101	10	1
3	010	010	110	12	1
4	110	110	010	4	2
5	001	101	111	14	2
6	100	000	001	2	2
7	011	010	110	12	1
8	100	101	111	14	2
9	111	111	110	12	1
10	111	111	101	10	1
11	011	010	110	12	1
12	100	101	100	8	1
13	001	101	100	8	1
14	100	000	010	4	2
15	101	000	100	8	1
16	010	011	111	14	2
17	101	001	101	10	1
18	000	100	101	10	1
19	110	110	010	4	2
20	000	000	010	4	2

$$\text{Average Fitness} = 174/20 = 8.07 \approx 8$$

$$\omega_{c2} = 1 - (11/20) = 9/20 = 0.45$$

$$U/\text{ind}(C-C_2) = \{\{1,5,13\}, \{2,15\}, \{3,16\}, \{4,19\}, \{6,8,12,14\}, \{7,11\}, \{9,10\}, \{17\}, \{18,20\}\}$$

$$\text{Pos}_{\{C-C_2\}}(D) = \{2,3,7,9,10,11,12,13,15,17,18\}$$

TABLE 5. REMOVING C_3

S.no	C - C_3	Crossover	Mutation (C)	Fitness	Group (D)
1	001	101	001	2	2
2	101	001	011	3	2
3	000	000	001	2	2
4	100	100	000	0	2
5	011	111	101	10	1
6	110	010	011	3	2
7	011	010	110	9	1
8	100	101	111	11	1
9	111	111	110	9	1
10	101	101	001	2	2
11	101	101	111	11	1
12	001	001	000	0	2
13	011	111	011	3	2

14	100	000	010	1	2
15	101	100	101	10	1
16	010	011	111	11	1
17	101	001	011	3	2
18	010	110	111	11	1
19	100	100	000	0	2
20	010	010	000	0	2

Average Fitness = $101/20 = 5.05 \approx 5$

$\omega_{c3} = 1 - (8/20) = 12/20 = 0.6$

$U/\text{ind}(C-C_3) = \{\{1,12\}, \{2,10,11,15,17\}, \{3\}, \{4,8,14,19\}, \{5,7,13\}, \{6\}, \{8,14,19\}, \{9\}, \{10,11,15,17\}, \{16,18,20\}\}$

$\text{Pos}_{\{C-C_3\}}(D) = \{5,7,8,9,11,15,16,18\}$

TABLE 6. REMOVING C_4

S.no	C - C_4	Crossover	Mutation	Fitness	Group
1	000	100	000	0	2
2	100	000	010	1	2
3	001	001	000	0	2
4	101	101	001	4	1
5	010	110	100	8	1
6	111	011	010	1	2
7	011	010	110	9	1
8	100	101	111	13	2
9	111	111	110	9	1
10	101	101	001	4	1
11	001	000	010	1	2
12	110	111	110	9	1
13	010	110	010	1	2
14	100	000	010	1	2
15	100	101	100	8	1
16	011	010	110	9	1
17	100	000	010	1	2
18	010	110	111	13	2
19	101	100	000	0	2
20	010	011	001	4	1

Average Fitness = $96/20 = 4.8 \approx 4$

$\omega_{c4} = 1 - (9/20) = 11/20 = 0.55$

$U/\text{ind}(C-C_4) = \{\{1\}, \{2,8,14,15,17\}, \{3,11\}, \{4,10,19\}, \{5,13,18,20\}, \{6,9\}, \{7,16\}, \{12\}\}$

$\text{Pos}_{\{C-C_4\}}(D) = \{4,5,7,9,10,12,15,16,20\}$

We know that keeping the most important attributes (genes) as 1 is a good choice. The most important attribute is C_1 . Keeping $C_1=1$ gives the chromosome 1000, which is a solution. C_3 is the second most important attribute. Keeping $C_1=1$ and $C_3=1$ gives the chromosome 1010, which is the optimal solution. Hence, knowing which attribute is important reduces the search space and helps to find the optimal solution.

VI. IMPLEMENTATION

Figure 2 shows the snapshot of the application designed to implement the hybrid algorithm. The application works for 4 items in the knapsack. It takes the weight and profit values of the items as the input and displays the optimal solution along with the items included in the knapsack.

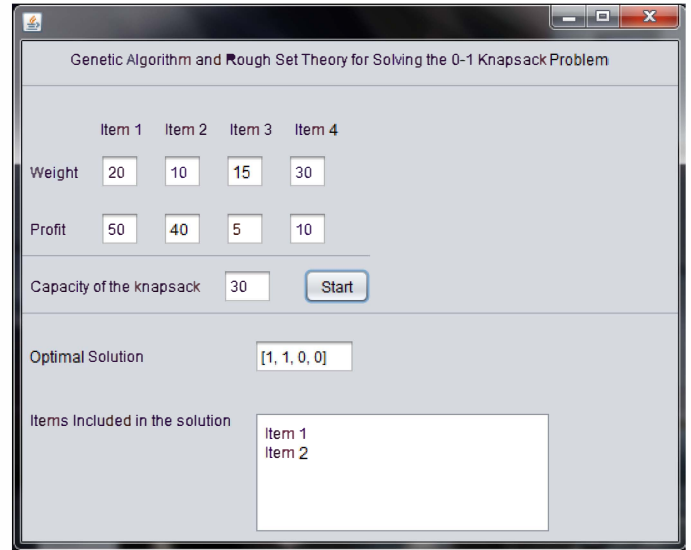


FIGURE 2. SNAPSHOT OF THE IMPLEMENTATION.

VII. 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.

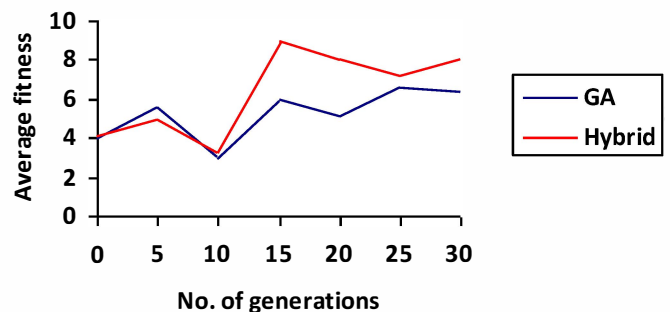


FIGURE 3. GRAPH FOR COMPARING GA AND THE HYBRID APPROACH

In Figure 3, the graph shows the comparison between two approaches: Solving the knapsack problem using Genetic

Algorithm (GA) indicated by the blue line, and the Genetic Algorithm and Rough Set Theory Hybrid algorithm, indicated by the red line. The y-axis shows the average fitness value of the chromosomes in one generation and the x-axis shows the number of generations. As shown in the figure, the hybrid algorithm produced the optimal solution in the 15th generation whereas the genetic algorithm did the same in the 25th generation. Also, the hybrid approach produced a better solution than the genetic algorithm, as indicated by the higher fitness values achieved by the hybrid approach.

TABLE 7. COMPARISON BETWEEN GA AND THE HYBRID APPROACH

	Genetic Algorithm	Hybrid Algorithm
Space Complexity	Relatively high	Relatively low (search space is reduced.)
Time Complexity	Relatively high (slower)	Relatively low (faster)
Efficiency	Not efficient for large no. of items in the knapsack.	Efficient for large no. of items in the knapsack.
Optimality	May not always give an optimal solution.	Always gives an optimal solution.

REFERENCES

- [1] Ritika Mahajan and Sarvesh Chopra. Analysis of 0/1 knapsack problem using deterministic and probabilistic techniques. In Proc. IEEE International Conference on Advanced Computing and Communication Technologies (ACCT'12), pages 150–155, Rohtak, Haryana, January 2012.
- [2] R.P. Singh. Solving the 0-1 knapsack problem using genetic algorithm. In Proc. International conference on Communication software and Networks (ICCSN'11), pages 591–595, Xian, China, May 2011.
- [3] P.G. Tharanipriya and P. Vishnuraja. Hybrid genetic algorithm for solving knapsack problem. In Proc. International conference on Information Communication and Embedded Systems (ICICES'13), pages 416–420, Chennai, India, February 2013.
- [4] D.L. Massart and B. Walczak. Rough set theory. In Proc. International conference on Chemometrics and intelligent systems, pages 1–16, Belgium, December 1998.
- [5] Han K H , Kim J H. Genetic quantum algorithm and its application to combinatorial optimization problem [C]. In Proc Proceedings of the 2000 Congress on Evolutionary Computation, Piscataway, 2000, 2: 1354 -1360.
- [6] Han K H , Park K H , Lee C H , et al. Parallel quantum-inspired genetic algorithm for combinatorial optimization problem[C]. In Proceedings of the 2001 IEEE Congress on Evolutionary Computation, 2001: 1422-1429.
- [7] S. Martello, D. Pisinger, P. Toth. New Trend in exact algorithms for the 0-1 knapsack problem, European Journal of Operational Research 123 (2000) 325–332.
- [8] D.E. Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning. (Addison-Wesley, New York, 1989).
- [9] J.H. Holland. Adaptation in Natural and Artificial Systems (University of Michigan Press, 1975; MIT, London, 1992).