International Journal of
Emerging Research in Management &Technology
ISSN: 2278-9359 (Volume-6, Issue-7)

Research Article | **July 2017**

Clustering Analysis of Greedy Heuristic Method in Zero_One Knapsack Problem

Dr. V. Selvi
Assistant Professor, Department of Computer Science, Mother Teresa Women's University, Kodaikanal, Tamil Nadu, India

Abstract-

Knapsack problem is a surely understood class of optimization problems, which tries to expand the profit of items in a knapsack without surpassing its capacity, Knapsack can be solved by several algorithms such like Greedy, dynamic programming, Branch & bound etc. The solution to the zero_one knapsack problem (KP) can be viewed as the result of a sequence of decision. Clustering is the process of resolving that type of applications. Different clustering application for grouping elements with equal priority. In this paper we are introducing greedy heuristic algorithm for solving zero_one knapsack problem. We will exhibit a relative investigation of the Greedy, dynamic programming, B&B and Genetic algorithms regarding of the complexity of time requirements, and the required programming efforts and compare the total value for each of them. Greedy and Genetic algorithms can be used to solve the 0-1 Knapsack problem within a reasonable time complexity. The worst-case time complexity (Big-O) of both algorithms is O(N). Using the greedy method, the algorithm can produce high quality clusters while reduce time the best partioning avoid the memory confinement problem during the process.

Key words- 0/1 Knapsack, Algorithm, Greedy algorithm, dynamic programming, Genetic.

I. INTRODUCTION

The 0-1 Knapsack Problem is vastly studied in importance of the real world applications that build depend it discovering the minimum inefficient approach to cut crude materials seating challenge of speculations and portfolios seating challenge of benefits for resource supported securitization, A few years ago the generalization of knapsack problem has been studied and many algorithms have been suggested. Advancement Approach for settling the multi-objective0-1 Knapsack Problem is one of them, and there is numerous genuine worked papers established in the writing around 0-1 Knapsack Problem and about the algorithms for solving them. The 0-1 KP is extremely well known and it shows up in the real life worlds with distinctive application. Knapsack problem in which numerous Knapsack are considered. The KP is a given an arrangement of items, each with weight and a value, decide the number of each item to include in a capacity so that the total weight is little than a given capacity and the total value must as large as possible. We have n of items. Each of them has a value Vi and a weight Wi. The most extreme weight that we can convey the knapsack is C. The 0 – 1 KP is an uncommon case of the original KP problem in which each item can't be Sub separated to fill a holder in which that input part fits. The 0 – 1 KP confines the quantity of each kind of item xj to 0 or 1. Mathematically the 0 – 1 KP can be formulated as:

Maximize $\sum^{n}_{i=1}$ PiXi subject to $\sum^{n}_{i=1}$ WiXi $\leq$C.

Assume 7 numbers of items arrive as shown in table 1. We need to choose such items so that it will satisfy our two goals as follows:

- Fill it to get the greatest benefit.
- Knapsack holds a most extreme of 22 pounds. So the aggregate weight of the chose items not surpasses our greatest limit.

In this research, a 0/1 KP is presented. As a solution of the 0/1 knapsack problem, greedy algorithm, dynamic programming algorithm, B&B algorithm, and Genetic algorithm are applied and evaluated both analytically and experimentally in terms of time and the total value for each of them, Moreover, a comparative study of the greedy ,dynamic programming, branch and bound, and Genetic algorithms is presented.

Table1:Knapsack Example

Items	1	2	3	4	5	6	7
Profit	10	8	9	15	7	7.2	5.5
Weight	12	8	6	16	4	5	8

II. PROPOSED WORK

In this section we will introduce the 0-1 knapsack problem, and we will present my related work of the algorithms used to solve knapsack problem and the comparisons done to illustrate the differences between them.

© www.ermt.net All Rights Reserved

A. 0/1 Knapsack problem (0/1 KP)

The first appears of knapsack problem was in 1957, in two publications. The first was a paper by George Dantzig (1957); He is a creator of the field of Operations Research and a developer of linear programming. He demonstrated that the persistent of the KP, The second paper is flawlessly maximized by selecting items by bang-for-buck.KP is a well-known optimization problem, which has restriction of the value either 0 (leave it) or 1 (take it), for a given collection of items, where each has a weight and a value, that to determine the items to be included in a sets, then the total cost is less or equal to a given capacity and the total profit is as max as possible. Obviously, the items are indivisible, accordingly the problem is been called "0-1 Knapsack Problem", that because you can't derive, that mean take all value of the item or leave it.

B. Greedy Algorithm and how to solve the problem

A greedy algorithm is an algorithmic paradigm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding global optimum. In many problems, a greedy strategy does not in general produce an optimal solution, but nonetheless a greedy heuristic may yield locally optimal solutions that approximate a global optimal solution in a reasonable time. This heuristic need not find a best solution, but terminates in a reasonable number of steps: finding an optimal solution typically requires unreasonable many steps. In mathematical optimization, greedy algorithms solve combinational problems having the properties of matroids.KP can be solved by many algorithms like Greedy algorithm by select the option that look like the best at the moment and its trust the local optimal solution will lead to a global optimal solution, Greedy are used for optimization problems. Its typically use some heuristic knowledge to create a pool of sub optimal that hope converges to an optimum solution

C. Dynamic Programming

Dynamic programming is typically applied to optimization problem. The word programming stands for planning and it does not mean by computer programming. Dynamic programming is a technique for solving problems with overlapping sub problem. In this method each sub problem is solved only once. The result of each sub problem is recorded in a table from which we can obtain a solution to the original problem. Characterize the structure of optimal solution that means a mathematical notation that can express any solution and sub solution for the given problem. The dynamic algorithm solve each sub problem individually, once the solution to a given sub problem has been computed, it will be stored in the memory, since the next time the same solution is needed, it's simply looked up. Distinctly, a Dynamic algorithm guarantees an optimal solution. Here are two key traits that the problem must have all together for dynamic programming to apply: the first one is an overlapping sub problems and the second is an optimal substructure.

D. Branch and Bound Algorithm

Branch and bound (BB, B&B, or BnB) is an algorithm design paradigm for discrete and combinatorial optimization problems, as well as mathematical optimization. A branch-and-bound algorithm consists of a systematic enumeration of candidate solutions by means of state space search: the set of candidate solutions is thought of as forming a rooted tree with the full set at the root. The algorithm explores branches of this tree, which represent subsets of the solution set. Before enumerating the candidate solutions of a branch, the branch is checked against upper and lower estimated bounds on the optimal solution, and is discarded if it cannot produce a better solution than the best one found so far by the algorithm. In fact, Branch & bound is a well-known technique that is mainly used to solve the problem which categorized as optimization problems. Despite the fact that, in the worst case still has an exponential complexity, but it is may use to solve a large cases of difficult mixed problems. Branch & Bound algorithm for the KP by exhibited which can acquire either optimal or inexact solutions. A few attributes of the algorithm are talked about and computational experience is introduced. And the B& B is a credulous way to deal the 0–1 KP is to consider thusly all the 2n possible solutions X, figuring the benefit every time and monitoring the most elevated benefit discovered and the relating vector.

E. Genetic Algorithm

Genetic Algorithms (GAs) Genetic Algorithms are computer algorithms that search for good solutions to a problem from among a large number of possible solutions. These computational paradigms were inspired by the mechanics of natural evolution, including survival of the fittest, reproduction, and mutation GAs is search algorithms that follow the concept of natural selection and genetics. Genetic Algorithms are based on the principle of heredity and evolution which claims "in each generation the stronger individual survives and the weaker dies". Therefore, each new generation would contain stronger (fitter) individuals in contrast to its ancestors. The process of GA's is iteration based of constant population size of candidate solutions. In each generation/iteration each chromosome's fitness in the current population is evaluated and new population evolves. Chromosomes with higher fitness values goes through reproduction phase in which selection, crossover and mutation operators are applied to get new population. Chromosomes with lower fitness values are discarded. Again this generated new population is evaluated and selection, crossover, mutation operators are applied. This process continues until we get an optimal solution for the given problem.

III. ALGORITHMS FOR SOLVING 0/1 KNAPSACK PROBLEM

In this section the Greedy, dynamic programming, B&B and Genetic algorithms will be presented.

© www.ermt.net All Rights Reserved

A. Greedy algorithm

Greedy Algorithm for solving 0-1 knapsack problem is calculate the ratio, where a ratio between the inputs values and the inputs weights will be calculated and according to this value the next input will be chosen to fill the knapsack in a proper way. A greedy algorithm mainly tests all inputs according to some preconditions then arranges them in a proper order to maximize or minimize the value of the required solution.

1. Algorithm:

The algorithm for solving knapsack problem with greedy approach is as given below.

```
Algorithm Knapsack_Greedy(w,n)
// P[i] contains the profit of i items
Such that 1≤i≤n.
// W[i] contains weights of I items.
// X[i] is the solution vector.
// W is the total size of knapsack.
For i:=1 to n do
{
If [w[i]<w] then // capacity of knapsack is a constraint
{
X[i]:=1.0;
W=W-w[i];
}
}
If(i<n) then x[i]:=W/w[i];
}
```

B. Dynamic Programming Algorithm

Dynamic programming is typically applied to optimization problem. The word programming stands for planning and it does not mean by computer programming. Dynamic programming is a technique for solving problems with overlapping sub problem.In this method each sub problem is solved only once. The result of each sub problem is recorded in a table from which we can obtain a solution to the original problem. Characterize the structure of optimal solution that means a mathematical notation that can express any solution and sub solution for the given problem.

1. Algorithm

There are two algorithms one for computing the matrix chain multiplication for optimum valued multiplication sequence. Another algorithm is to extract the optimum sequence.

```
Algorithm Matrix_Chain_Mul(arrayP[1.....n]
{
// We can maintain a parallel array s[i,j] in which we will store the value of providing the optimal split such that array
// s[1....n-1 , 2.....n]
K is a  optimal spit such that for Ai.....j first multiply the
// sub chain Ai :k and then // multiply the sub chain Ak+1 , j
// len is length of sub chain.
For i:=1 to n do
M[i,i] : =0; // initialize
For len :=2 to n do
{
For i :=1 to (n-len+1) do
{
j: =i+len-1;
M[i,j]: =∞;
For k: =i to j-1 do
{
// check all splits
q: =M[i,k]+M[k+1,j]+P[i-1]*P[k]*P[j]
if(q<m[i,j])
{
M[i,j]:=q;
S[i,j]:=k;
}
}
```

© www.ermt.net All Rights Reserved

```
}
}
return M[1,n]; // optimum value
}
```

For extracting optimum sequence we will use following algorithm.

```
Algorithm Mul[i,j]
// P=A[i].....A[k]
// Q=A[k+1].....A[j]
{
if (i==j) then
return A[i];
else
{
K:=S[i,j]
P:=Mul(i,k)
Q:=Mul(k+1,j)
return P*Q;
// multiply matrices
P and Q
}
}
```

C. Branch and Bound Algorithm

This section presents the branch & bound Algorithm for solving the 0-1 knapsack problem .branch & bound is a technique that is used to solve the problems that categorized as optimization problems. It is a changeover comprehensive search, on the grounds that not at all like it, branch & bound builds hopeful arrangements one part at a time and assesses the somewhat developed solutions. On the off chance that no potential estimations of the remaining parts can prompt a solution, the remaining segments are not created. This methodology makes it conceivable to settle some huge occasions of troublesome difficult combinatorial problems, however, in the most pessimistic scenario; regardless it has an exponential complexity. B & B is based on state space tree. The state space tree is a root of the tree where every level represent to a decision in the solution space that relies on the upper level and any conceivable solution is represented to by a few ways beginning at the root and finishing with a leaf.

1. Algorithm:

```
Branch & Bound Pseudo code
Input:
Array of Weights and array of values
Output:
Max Value
Note: Items are sorted according to value/weight ratios
Queue Q
Node Type: current, temporary
*Create the root
Q.enqueue(root)
Max Value = value
While (Q is not empty)
current = PQ.GetMax()
if (current >MaxValue)
Then Set the left child of the current node to include the next item8
If child.Left value is greater than MaxValue
MaxValue = Value of the Left Child
End if
If child.left bound better than MaxValue
Q.enqueue(Left Child)
End if
If child.Right bound better than MaxValue
Q.enqueue(Right Child)
End if
Return Best solution
```

© www.ermt.net All Rights Reserved

D. Genetic Algorithm

Genetic Algorithms which is computer algorithm that looks for good solution for a problem from among huge arrangements of possible solutions. They were proposed and developed in the 1960s by John Holland, his students, and his colleagues at the University of Michigan. These computational paradigms were inspired by the mechanics of natural evolution, including survival of the fittest, reproduction, and mutation. These mechanics are well suited to resolve a variety of practical problems, including computational problems, in many fields. Some applications of GAs are optimization, automatic programming, machine learning, economics, immune systems, population genetic, and social system. The main idea of Gas an arrangement of applicant solutions (chromosomes) called population. A new population is generated from an old population in any expectation of getting a better Solution.

*1. Selection Method***:**

Table2: Selection method

Items	0	1	2	3	4	5	6	7	8	9	10	11
Chr fitness	40	20	5	1	9	7	38	27	16	19	11	3
Indexes	0	1	2	3	4	5	6	7	8	9	10	11

2. Fitness Array

Table3:Fitness Array

Items	0	1	2	3	4	5	6	7	8	9	10	11
Indexes	0	6	7	1	9	8	10	4	5	2	11	3

IV. ANALYTICAL METHOD

The target of any algorithm solving KP is to perform productive efficient solution in the minimum possible time.

1) Greedy algorithm

The complexity time for greedy algorithm execution time will be as:

1. Sorting by Merge sort algorithm is O(NlogN)
2. $\Sigma 1=n-0$ ni=0 is O(N)

From 1 and 2, the total complexity is O (NlogN) + O(n) which approximately equal O(NlogN).

2) Dynamic programming algorithm

The worst case time complexity of the dynamic programming algorithm used to solve the 0-1 KP is O ($W*n$)

3) Branch and bound algorithm

In the worst case, the B&B algorithm will generate all intermediate stages and all leaves. Therefore, the tree will be complete then the Time complexity = O (2n).

4) Genetic algorithm

The function for introduces the array chromosomes has an O (N). Crossover, fitness and mutation functions have O (N). The two selection functions have O (1).The function that checks for the terminating condition has O (1). Then the total complexity of the program is O (N).

Table4: Time Complexity

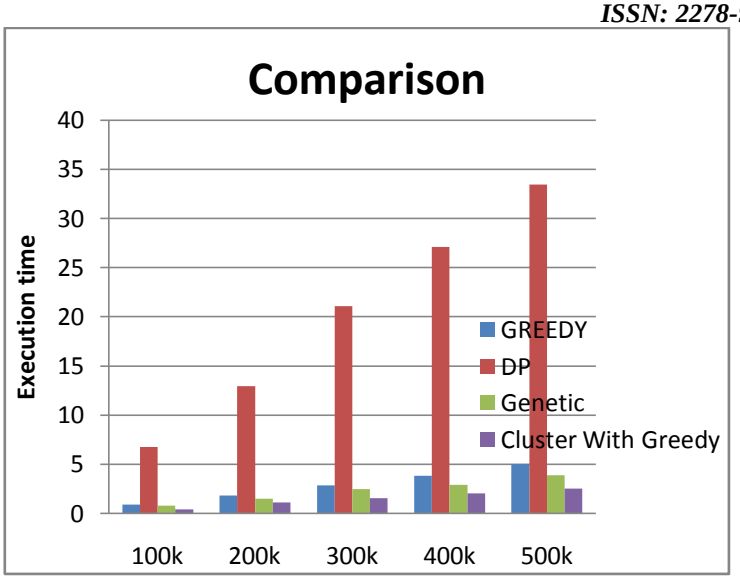
Metric	Greedy	Dynamic programming	Branch & Bound	Genetic	Cluster Greedy
EX.Time	O(NlogN)	O(W*N)	$O(2^n)$	O(N)	O(N)

A. Experimental Result

The version of all algorithms presented (Greedy, dynamic programming, Branch and bound and Genetic Algorithm) has been coded in C. The experimental time that appears in table 16 is the execution time for Greedy, DP, B&B and Genetic Algorithm with different size where K mean thousand which 100K is mean 100,000 items. Since the Branch and bound $O(2^n)$ then its need more space and in the device that we are test on, it does not work for 100000 array size so we test it just to the max number(60000) and Capacity size(100) that what we can and the experimental time for the four algorithms. From the result we can see that all of the experimental time for each algorithm are expected depend on the analytical model ,we can see that the minimal time is for genetic algorithm then Greedy, DB and B&B respectively. The dynamic programming algorithm are always give the optimal result but the greedy and genetic algorithms are given the local optimal result, for that we are implement each of them on the same data set to compare which one that give the best local optimal result.

Size	Greedy	DP	Genetic	Cluster greedy
100k	0.8623	6.7177	0.7825	0.3623
200k	1.7758	12.9441	1.4869	1.0758
300k	2.8489	21.0783	2.4652	1.5489
400k	3.8071	27.1098	2.8836	2.0071
500k	4.9852	33.5011	3.8795	2.4852

© www.ermt.net All Rights Reserved



V. CONCLUSION

The greedy, dynamic programming, branch and bound and genetic algorithms have been presented. The performed analysis and the conducted comparisons have been presented, and compared to the experiment results obtained from applying these algorithms on 0/1 knapsack problem. As Greedy algorithm are generally fast. Greedy method is often not suited for large problem due to lack of parallelism. The worst execution time is suffered by the branch and bound algorithm, since its complexity grows exponentially. The best execution time is suffered by genetic and Cluster Greedy algorithms since its complexity grows are O (N).

REFERENCES

[1] T.H.Cormen, C.E.Leiserson and R.L.Rivest, *Introduction to algorithms*, I IT Press, Cambridge MA,1996.
[2] Levitin, Anany. *The Design and Analysis of Algorithms*. New Jersey: Pearson Education Inc., 2003.
[3] *Different Approaches to Solve the 0/1 Knapsack Problem*. Maya Hristakeva, Dipti Shrestha; Simpson Colleges
[4] Mohanty, R. Satapathy, "An evolutionary multiobjective genetic algorithm to solve 0/1 Knapsack Problem," IEEE Transl. Beijing, vol. 2, pp. 397–399, August 2009. S.
[5] Springer, NJ. USA, vol. 7, pp. 16–28, August 2007. M. Babaiof, M. Babaiof, D. Kempe "A Knapsack Secretary Problem with Applications,"
[6] M. Hristakeva, D. Shrestha, "Solving the 0-1 Knapsack Problem with Genetic Algorithms," IEEE Transl. Beijing, Science & Math Undergraduate Research Symposium, Indianola, Iowa, Simpson College June 2004.
[7] Springer. Ch2, Georgia Institute of Technology, 2010. J. Bartholdi, "The Knapsack Problem,"

© www.ermt.net All Rights Reserved