International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2018 IJSRCSEIT | Volume 3 | Issue 1 | ISSN : 2456-3307

An Overview of Combinatorial Optimization Techniques

Dr. S. Dhanalakshmi
Professor, Department of Computer Science and Engineering, Malla Reddy Engineering College (A), Secunderabad, Telangana, India

ABSTRACT

This paper presents the overview of greedy technique, dynamic programming technique and branch & bound technique. Knapsack problem is one of the applications of that technique. This discussion is centered overview of capacity of the objects and then the objective is to obtain a filling of the knapsack that maximizes the total profit earned without exceeding the capacity of the knapsack. After solving, the problems with maximum profit then find the time complexity of greedy, dynamic programming and branch & bound d techniques

Keywords: 0/1 or fractional knapsack, Capacity, G reedy, Dynamic Programming and Branch & Bound

I. INTRODUCTION

Knapsack Problem

Knapsack problem is a problem in combinatorial optimization; given a set of items, each with a weight and a value, we have to pack the knapsack with maximum value in such a manner that the total weight of the items should not be greater than the capacity of the knapsack. The knapsack problem can be categorized into 0/1 knapsack problem or binary knapsack problem (in 0/1 each item may be taken 1 or not 0) and fractional knapsack problem (bounded knapsack problem and unbounded knapsack problem). Dynamic Programming and Branch & Bound algorithm can works with 0/1 knapsack problem and greedy algorithms does not work with 0/1 knapsack problem, it work with fractional knapsack problem.

II. RELATED WORK

Overview of Greedy, Dynamic Programming and Branch & Bound : Dynamic Programming technique are bottom up approach, each step depends on the solution to sub problems, then use the solutions of those sub problems to make an optimal choice, it's based on 0/1 knapsack problem; Greedy Technique are top down approach, to make an optimal choice (without knowing solutions to sub problems) and then solve remaining sub problems, it's based on fractional knapsack problem; but both techniques are used for optimization techniques, and both build solutions from a weight and profit of individual elements but does not exceed the maximum capacity for satisfying the constraints. Branch and Bound algorithms refers to all state space search methods in which all children of the expanded node (E-node) are generated before any other live node. This algorithm consists of a systematic enumeration of candidate solutions, discarding large subsets of fruitless candidates by using upper and lower estimated bounds of quantity being optimized.

III. GREEDY ALGORITHMS

Greedy algorithm is a general design or most straight forward design technique; it's used for optimization problems. Simply choose best option at each step, steps for achieving greedy algorithms are feasible, local optimal choice and irrevocable/unalterable. It is based on two types of paradigms, one for subset and

another for ordering paradigm. Subset paradigms are 0/1 knapsack problem, minimum cost spanning trees, job sequencing with deadlines and ordering paradigms are single source shortest path problems, these are the applications of greedy algorithms.

Fractional Knapsack Problem in Greedy Solution

Fractional knapsack problem, we can break the items for maximizing the total value of knapsack. The items can be break into three lemmas, this problem is called fractional knapsack problem. Assume knapsack holds weight W and items have Value V_i and Weight W_i; Rank the items by Value/Weight ratio, considering the items in order of decreasing ratio. It is based on three lemma. In Lemma 1 - the items are arranged by their values/profits. Here the items with maximum value is selected first and process continue with minimum value, Lemma 2 - The items are arranged by their weights, light weights is selected first and process continue with maximum weight and Lemma 3 - the items are arranged by certain ratio of profit/weights, here selection proceeds from maximum ratio to minimum ratio and then the objects are arranged in non decreasing order of Pi/Wi.

Algorithm:

```
Greedy knapsack (m,n)
// Profits & Weights of P[1…n] and W[1…n]
// P[i] / W[i]>= P[i+1] /W[i+1]
// m is the knapsack size
{
For i= 1 to n do x[i] = 0.0
U=m
For j = 1 to n do
{ if(w[i]>U) then break
X[i] = 1.0; U=U-w[i]
} if(i<=n) then x[i] = U/w[i]
}
```

Example: n=3, P= (25, 24, 15), W= (18, 15, 10) and m=20

Lemma	A	B	C	Weight	Profit
				(Wi)	(Pi)
Maximum Profit	25	2/15	0	18+2=20	25+(24/15*2)=28.2
Minimum Weight	0	10/15	10	10+10=20	15+(24/15*10)=31
Profit/Weight	0	24	5/10	15+5=20	24+(15/10*5)=31.5

In Lemma 1, object 1 has the largest profit value, so it is placed in to the knapsack first, then X1 has a profit of 25 is earned, weight of 18 and two units of knapsack capacity are left, so the maximum profit is 28.2. Similarly, in lemma, 2 are minimum weight the capacity is 31; finally the lemma 3 are profit by weight and the Maximum Profit for the knapsack problem in greedy method is 31.5.

Fractional Knapsack has time complexity O (N log N) where N is the number of items.

IV. DYNAMIC PROGRAMMING

Dynamic Programming is also called dynamic optimization; it is a method for solving a complex problem into simpler sub problems. The next time the same sub problem occurs, instead of recomputing its solution, one simply looks up the previously computed solution.

There are three basic elements that characterize a dynamic programming algorithm;

- Substructure – decompose the given problem into smaller sub problems;
- Table structure – after solving the sub problems, store the answers to the sub problems in a table
- Bottom-up construction – using table, combine solutions of smaller sub problems to solve larger sub problems, and eventually arrive at a solution to the complete problem.

Steps:

1. $S^0 = \{(0,0)\}$ then we compute S^{i+1} from S^i by computing i=0,1,2,---,n
2. $S_1^i = \{(P,W) / P-P_{i+1}, W-w_{i+1}) £ S^i\}$
3. Now S^{i+1} can be computed by merging the pairs in S^i and S_1^i together

4. Note that the S^{i+1} contains two pairs (P_j,W_j) and (P_k,W_k) with the property that $P_j \le P_k$ and $W_j \ge W_k$, then the pair (P_j,W_j) can be discarded.

5. A discarded or purging rule is called dominance rules. In purging rules basically the pair with less profit and more weights.

Algorithm:

```
Dynamic knapsack (p,w,n,m)
{
S^0 ={(0,0)};
for i=1 to n-1 do
{
S^i-1= {(P, W) | (P-pi, W-wi) £ S^i-1 and W ≤m};
S^i = Merge Purge (S^i-1,S1^i-1);
}
(PX, WX) = last pair in S^n-1;
(PY, WY) = (P^1 + pn, W^1+wn) where W^1 is the largest W in
Any pair in S^n-1 such that W + wn ≤ m;
If (PX>PY) then xn=0;
Else xn=1;
}
```

Example:

Consider the knapsack instance m=6, (w1,w2,w3)=(2,3,4) and (p1,p2,p3)=(1,2,5)

$S_0 = \{(0,0)\}$

$S_1^0 = \{(1,2)\}$

S^{i+1} by merging the pairs of S_i and S_1^i

$S^1 = \{(0,0),(1,2)\}$

$S_2^1 = \{(2,3),(3,5)\}$

S^2 = merge S^1 and S_2^1 = {(0,0),(1,2), (2,3),(3,5)}

(P_j,W_j) and (P_k,W_k) with the property that $P_j \le P_k$ and $W_j \ge W_k$, then the pair (P_j,W_j) can be discarded, (3,5) is discarded according to purging rules or dominance rules.

$S_1^2 = \{(5,4),(6,6),(7,7),(8,9)\}$

$S^3 = S^2$ and S_1^2 = {(0,0),(1,2), (2,3), (5,4),(6,6),(7,7),(8,9)}

Now we have a pair of (6,6)=(P,W)

X = {0,0,0} , so X_3 as 1={0,0,1}

We select next object (P-P_3) and (W-W_3)

ie.,(6-5) (6-4) = (1,2) so X_1 as 1={1,0,1}

We select next object (P-P_1) and (W-W_1)

ie.,(1-1) (2-2) = (0,0)

Total Profit earned =

$P_1X_1 + P_2X_2 + P_3X_3 = 1.1 + 2.0 + 5.1 = 6$

Total Weight earned =

$W_1X_1 + W_2X_2 + W_3X_3 = 2.1 + 3.0 + 4.1 = 6$

The objects 1 and 3 are selected based on the knapsack instance. They give maximum profits and same weights not exceed 6.

V. BRANCH AND BOUND

Branch & Bound is general algorithm or Systematic method for finding optimal solution of various optimization problems, especially in discrete and combinatorial optimization. The B&B strategy is very similar to backtracking in that a state space tree is used to solve a problem. B&B refers to all state space search methods in which all children of the "E-node" are generated before any other "live node" can become the "E-node". Live node is a node that has been generated but whose children have not yet been generated. E-node is a live node whose children are currently being explored. Dead node is a generated node that is not to be expanded or explored any further. All children of a dead node have already been expanded. Both BFS & D-search (DFS) generalized to B&B strategies. BFS like state space search will be called FIFO (First in First Out) search, as the list of live nodes is "First-in-first-out" list or queue. D-search (DFS) like state space search will be called LIFO (Last in First Out) search as the list of live nodes is a "last-in-first-out" list (or stack).

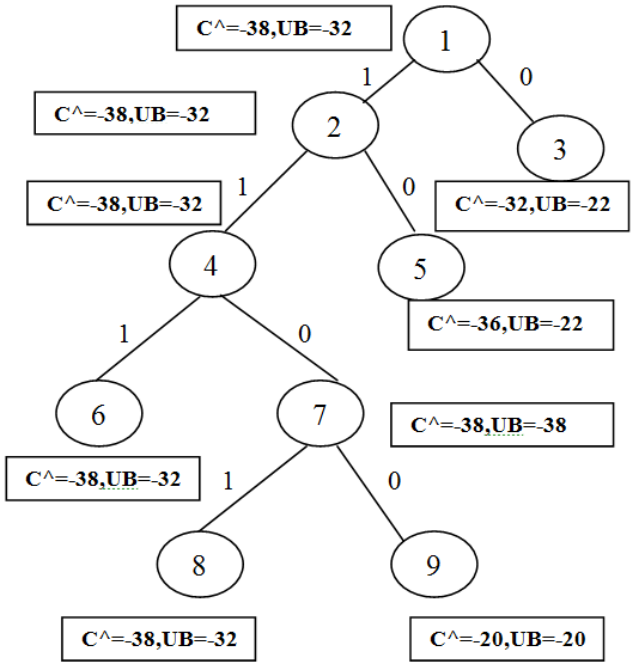
Algorithm:

```
UBound (cp,cw,k,m)
// cp, cw, k and m have the weight and profit of the i^th object
{
b=cp;
c=cw;
{
If(c+w[i]≤m) then
{
c=c+w[i];
b=b-p[i];
} }
```

return b;
}

Example:

Consider the knapsack instance n=4, (p1,p2,p3,p4)=(10,10,12,18), w1,w2,w3,w4)=(2,4,6,9) and m=15.



The object x1=1,x2=1,x3=0 and x4=1 are selected weight of the objects are x1=2,x2=4,x3=0 and x4=9 Total weights are 2+4+0+9= 15 selected

X = (1101)

VI. CONCLUSION

In this paper, we have presented the overview of different combinatorial optimization techniques of greedy method, backtracking algorithms and branch & bound algorithm. Its depends on the capacity of knapsack and size of the population, it can be useful for analysis of genetic algorithm approach, In proposed method, to implement some of the more advanced approximation schemes and compare with their performance of genetic algorithm paradigms.

VII. REFERENCES

[1]. S. P. Sajjan, Ravi kumar Roogi, Vijay kumar Badiger, Sharanu Amaragatti,"A New Approach to Solve Knapsack Problem", An International Research Journal of Computer Science and Technology, ISSN:0974-6471 OnlineISSN: 2320-8481,Vol7,Issue2, {http://www.computerscijournal.org/vol7no2/a-new-approach-to-solve-knapsack-problem/pdf/vol7no2/vol7no2_219-222.pdf}

[2]. Veenu Yadav1, Ms.Shikha Singh, Vijay kumar Badiger, Sharanu Amaragatti," A Review Paper on Solving 0-1 knapsack Problem with Genetic Algorithms",International Journal of Computer Science and Information Technologies, Vol. 7 Issue 2, 2016, PP 830-832 ISSN : 0975-9646. { http://ijcsit.com/docs/Volume%207/vol7issue2/ijcsit2016070286.pdf}

[3]. Ameen Shaheen, Azzam Sleit, "Comparing between different approaches to solve the 0/1 Knapsack problem", IJCSNS International Journal of Computer Science and Network Security, ISSN:1738–7906,Vol.16 No.7,July 2016,PP1-10, { http://paper.ijcsns.org/07_book/201607/20160701.pdf}

[4]. Shafiqul Abidin, "Greedy Approach for Optimizing 0-1 Knapsack Problem", Communications on Applied Electronic, Vol 7 IssueNo.6, September 2017, ISSN : 2394-4714,PP1-3, {http://www.caeaccess.org/archives/volume7/number6/abidin-2017-cae-652675.pdf}

[5]. Salem Hildebrandt & Christopher Hanson, "0-1 Knapsack Optimization with Branch-and-Bound Algorithm", {http://www.micsymposium.org/mics2016/Papers/MICS_2016_paper_42.pdf}