International Journal of Computer Applications Technology and Research
Volume 8–Issue 08, 327-330, 2019, ISSN:-2319–8656
DOI:10.7753/IJCATR0808.1007

Comparative Study of Dynamic Programming and Greedy Method

San Lin Aung
Information Technology, Supporting and Maintenance Department
University of Computer Studies (Mandalay)
Myanmar

Abstract: This paper discusses relationships between two approaches to optimal solution to problems: Greedy algorithm and dynamic programming. Greedy algorithm has a local choice of the sub-problems whereas Dynamic programming would solve the all sub-problems and then select one that would lead to an optimal solution. Greedy algorithm takes decision in one time whereas Dynamic programming takes decision at every stage. This paper discuss about Dynamic Programming and Greedy Algorithm to solve the Knapsack Problem where one has to maximize the benefit of items in a knapsack without extending its capacity. The paper discusses the comparison of each algorithm in terms of time different Item values. The comparison result of two algorithms are described with the best local result produced by the algorithm against the real exact optimal result.

Keywords: Greedy Algorithm, Dynamic Programming, 0/1 Knapsack, Algorithm

1. INTRODUCTION

The Knapsack problem is a combinatorial optimization problem where one has to maximize the benefit of objects in a knapsack without exceeding its capacity. The objective of the paper is to present a comparative study of the dynamic programming, and greedy algorithms. The 0-1 Knapsack Problem is vastly studied in importance of the real world applications. 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.

Greedy strategy is to point out the initial state of the problem, through each of the greedy choice and get the optimal value of a problem solving method [8]. Dynamic programming is an optimization approach that transforms a complex problem into a sequence of simpler problems; its essential characteristic is the multistage nature of the optimization procedure. More so than the optimization techniques described previously, dynamic programming provides a general framework for analyzing many problem types. Within this framework a variety of optimization techniques can be employed to solve particular aspects of a more general formulation [2].

The paper is organized as follows: section 2 describes The Knapsack Problem (KP). Section 3 expresses Dynamic Programming and Greedy algorithm. This section also includes example of dynamic algorithm and the basic idea of the greedy algorithm. Section 4 presents Experimental Results and Section 5 illustrates the Conclusion.

2. THE KNAPSACK PROBLEM (KP)

The Knapsack Problem is an example of a combinatorial optimization problem, which seeks for a best solution from among many other solutions. Given a set of items, each with a mass and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. We have n of items. Each of them has a value Vi and a weight Wi.

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 [1]. 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. 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: [1, 3, 4]

$$\text{Maximize} \sum_{i=1}^{n} \mathbf{Pi\ Xi} \text{ Subject to } \sum_{i=1}^{n} \mathbf{Wi\ Xi} \leq C$$

International Journal of Computer Applications Technology and Research
Volume 8–Issue 08, 327-330, 2019, ISSN:-2319–8656
DOI:10.7753/IJCATR0808.1007

3. DYNAMIC PROGRAMMING AND GREEDY ALGORITHM

Dynamic algorithm is an algorithm design method, which can be used, when the problem breaks down into simpler sub problems; it solves problems that display the properties of overlapping sub problems. In general, to solve a problem, it's solved each sub problems individually, then join all of the sub solutions to get an optimal solution [2, 5].

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.

Dynamic Programming is a technique for solving problems whose solutions satisfy recurrence relations with overlapping subproblems. Dynamic Programming solves each of the smaller subproblems only once and records the results in a table rather than solving overlapping subproblems over and over again. To design a dynamic programming algorithm for the 0/1 Knapsack problem, we first need to derive a recurrence relation that expresses a solution to an instance of the knapsack problem in terms of solutions to its smaller instances. Consider an instance of the problem defined by the first i items, 1<= i<= N, with:

weights w1, ... , wi,
values v1, ... , vi,
and knapsack capacity j, 1<= j<= Capacity.

3.1 Algorithm Dynamic Programming

ALGORITHM Dynamic Programming
(Weights [1 ... N], Values [1 ... N], Table [0 ... N, 0 ... Capacity])
//Input: Array Weights contains the weights of all items
Array Values contains the values of all items
Array Table is initialized with 0s; it is used to store the results from the dynamic programming algorithm.
// Output: The last value of array Table (Table [N, Capacity])
contains the optimal solution of the problem for the given Capacity
for i = 0 to N do
for j = 0 to Capacity
if j < Weights[i] then Table[i, j] ←Table[i-1, j]
else Table[i, j] ← maximum { Table[i-1, j] AND Values[i] + Table[i-1, j – Weights[i]]
return Table [N, Capacity]

3.2 An Elementary Example

In order to introduce the dynamic-programming approach to solving multistage problems, in this section we analyze a simple example. Figure 1 represents a street map connecting homes and downtown parking lots for a group of commuters in a model city. The arcs correspond to streets and the nodes correspond to intersections. The network has been designed in a diamond pattern so that every commuter must traverse five streets in driving from home to downtown. The design characteristics and traffic pattern are such that the total time spent by any commuter between intersections is independent of the route taken. However, substantial delays, are experienced by the commuters in the intersections. The lengths of these delays in minutes, are indicated by the numbers within the nodes. We would like to minimize the total delay any commuter can incur in the intersections while driving from his home to downtown. [6, 7]

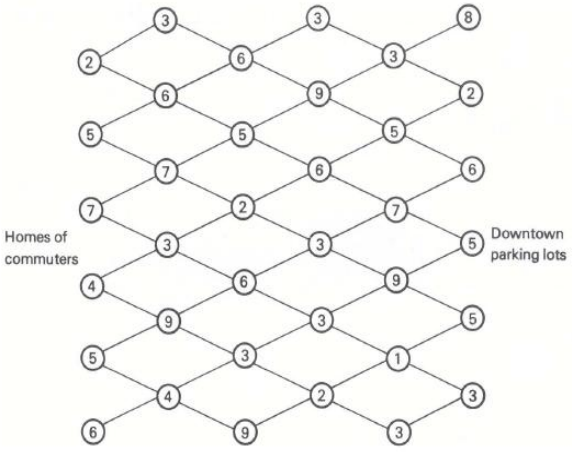

Figure1. Street map with intersection delays.

Figure 1 provides a compact tabular representation for the problem that is convenient for discussing its solution by dynamic programming. In this figure, boxes correspond to intersections in the network. In going from home to downtown, any commuter must move from left to right through this diagram, moving at each stage only to an adjacent box in the next column to the right. We will refer to the ‘‘stages to go," meaning the number of intersections left to traverse, not counting the intersection that the commuter is currently in.

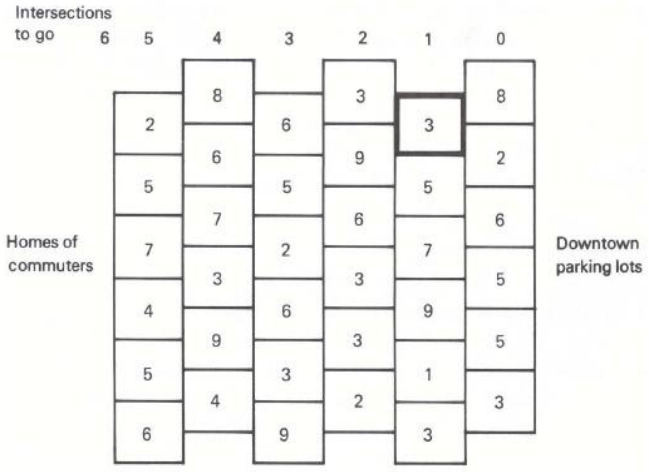

Figure 2. Compact representation of the network.

International Journal of Computer Applications Technology and Research
Volume 8–Issue 08, 327-330, 2019, ISSN:-2319–8656
DOI:10.7753/IJCATR0808.1007

The most naive approach to solving the problem would be to enumerate all 150 paths through the diagram, selecting the path that gives the smallest delay. Dynamic programming reduces the number of computations by moving systematically from one side to the other, building the best solution as it goes. Suppose that we move backward through the diagram from right to left. If we are in any intersection (box) with no further intersections to go, we have no decision to make and simply incur the delay corresponding to that intersection. The last column in Figure 2 summarizes the delays with no (zero) intersections to go [6, 7].

3.3 Greedy algorithm

A greedy algorithm is a straight forward design technique, which can be used in much kind of problems. Mainly, a greedy algorithm is used to make a greedy decision, which leads to a feasible solution that is maybe an optimal solution. Clearly, a greedy algorithm can be applied on problems those have 'N' number of inputs and we have to choose a subset of these input values those satisfy some preconditions.

Where, the next input will be chosen if it is the most input that satisfies the preconditions with minimizes or maximizes the value needed in the preconditions. A greedy algorithm is a straight forward design technique, which can be used in much kind of problems. Mainly, a greedy algorithm is used to make a greedy decision, which leads to a feasible solution that is maybe an optimal solution [4, 8].

3.4 The basic idea of the greedy algorithm

Greedy algorithm is a step by step, according to a certain optimization measure; each step should be able to ensure that the local optimal solution can be obtained. If the next data and partial optimal solution is no longer feasible solution, then the data cannot be added to the partial optimal solution until all the data are enumerated or cannot be added so far.

The hierarchical processing method of the optimal solution which can be obtained by some kind of measure is called the greedy strategy. If you want to use the greedy strategy to solve the problem, it is necessary to solve the following two problems: [4, 8]

(1) Whether the problem can be solved by greedy strategy;

(2) How to determine the greedy choice strategy, to get the best or better solution.

3.5 Algorithm Greedy Algorithm

ALGORITHM GreedyAlgorithm (Weights [1 ... N], Values [1 ... N])
// Input: Array Weights contains the weights of all items
Array Values contains the values of all items
// Output: Array Solution which indicates the items are included in the knapsack ('1') or not ('0')
Integer CumWeight
Compute the value-to-weight ratios ri = vi / wi, i = 1, ..., N, for the items given Sort the items in non-increasing order of the value-to-weight ratios for all items do
if the current item on the list fits into the knapsack
then place it in the knapsack
else proceed to the next one

4. COMPARISON RESULTS

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.

For testing, different file sizes are generated integers representing the weight and value of each item. We are test all of them using different array size but with the same Capacity size on 50, 100, 200, and 500. Sample result of testing for capacity = 50 is as shown in table 1 and figure 3.

Table 1 Comparison for Capacity = 50

Number of Items	Total Value GA	Total Value DP
100	1360	1385
200	3100	3320
300	5400	5600
400	8250	8535
500	13500	13700

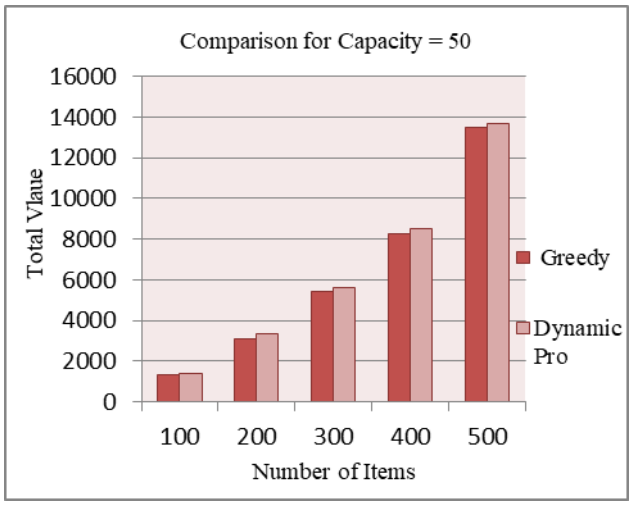

Figure 3 Comparison of Dynamic Programming and Greedy Algorithm

5. CONCLUSION

We can conclude that dynamic programming algorithms outperform and the greedy and genetic algorithm in term of the total value it generated. Greedy algorithm lacks with parallelism property whereas Dynamic Algorithm are exposed to parallelism. Both Greedy and Dynamic programming algorithm try to find out the optimal solution.

In both algorithm an optimal solution to the problem contains within it optimal solutions to sub-problems. Greedy method work efficiently for some problems like Minimum Spanning tree while it is not best suited for some problem like Travelling Sales man ,0/1 Knapsack. Dynamic method always generates optimal solution but they are less efficient than Greedy algorithm. As Greedy algorithm are generally fast.

Hence, paper presents a comparative study of the Greedy and dynamic methods. It also gives complexity of each algorithm with respect to time and space requirements. A feasible solution that does this is called an optimal solution. In order to solve a given problem, each problem has N inputs and requires finding a feasible solution that either maximizes or minimizes a given objective function that satisfies some constraints.

REFERENCES

[1] AuGallo, G.; Hammer, P. L.; Simeone, B. (1980). "Quadratic knapsack problems". Mathematical Programming Studies 12: 132–149.
[2] Kleywegt, D.Papastavrou, “The Dynamic and Stochastic knapsack Problem,” Opns. Res, pp. 17–35, 1998.
[3] 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.
[4] M.Lagoudakis, “The 0–1 knapsack problem—an introductory survey citeseer.nj.nec.com/151553.html, 1996.
[5] S. Martello, D. Pisinger, P. Toth “Dynamic Programming and Strong Bounds for the 0-1 Knapsack Problem,” Management Science, vol. 45, pp. 414–424, 1999
[6] http://en.wikipedia.org/wiki/Dynamic_programming
[7] http://www.geeksforgeeks.org/dynamic-programmingset-23-bellman-ford-algorithm/
[8] http://en.wikipedia.org/wiki/Greedy_algorithm