THINK INDIA JOURNAL
ISSN: 0971-1260
Vol-22-Issue-3-July-2019

Analytical Study of Greedy Algorithm To Solve Optimization Problems

Oshin
Lovely faculty of technology and sciences
School of computer science and engineering
Lovely Professional University.
Phagwara, Punjab, India

Surbhi
Lovely faculty of technology and sciences
School of computer science and engineering
Lovely Professional University. Phagwara,
Punjab,India

Abstract

Greedy algorithm is an intuitive and simple algorithm used for solving optimization problems. Algorithm works on sub-solutions with the intention to solve entire problem. At every step, it finds a local optimal solution with reference to current problem state and proceeds towards reaching the global optimal solution. The algorithm is not efficient to solve all optimization problems. Those problems that exhibit greedy choice property and optimal substructure can only be solved by greedy algorithm. This paper discusses few optimization problems that can be solved using greedy approach, along with the proof that how they qualify to be solved by the algorithm.

1. Introduction

Algorithms that are intended to reach optimal solution undergoes series of steps, at every step set of local choices are made. Greedy algorithm consistently makes local optimal choices that fit best at the moment with the belief that it may edge towards global optimal solution[1][2]. Widely, greedy algorithms are designed by going through the following steps:

i. From optimization problem, make a local optimal choice such that exactly one sub problem is left to be solved.
ii. Original problem is put to test by making greedy choices, such that it concludes to an optimal solution.

ISSN: 0971-1260
Vol-22-Issue-3-July-2019

iii. Make a greedy choice from original problem and prove that sub problem remained exhibit a unique trait that, if optimal solution subjected to it is conjoined with greedy choice then it generates an optimal solution to original problem.

Can we generate optimal solution to all optimization problems using greedy approach? Answer is "Big NO". Then how one can estimate that which of the specific problem could be guided by greedy heuristics? If a problem has following characteristics, then we can say that greedy choices will mark an optimal solution for it.

a. Greedy-choice property

This property states that global optimal solution is populated by making local optimal choices. At very decision point, a greedy choice is made irrespective of whether it will contribute to future solution. The algorithm follows top down approach and makes choices that may be calculated from past heuristics but had no relation with futuristic choices. We can pre-process input values or use some data structure to make quick greedy choices.

b. Optimal substructure

A problem is said to have optimal substructure if and only if optimal solution to an original problem comprises of optimal solution to sub problems. For a problem, we will make a greedy choice and we need to demonstrate that sub problem left have an optimal solution, and if it is united with greedy choice then it will yield a global optimal solution for the original problem.

Following sections exemplify some classic optimization problems that exercise greedy approach to reach global optimal solution. We will try to prove that problems qualify to have two main key components; hence they can be exploited by greedy algorithm.

1.1. Fractional Knapsack Problem

Thief planned to rob a jeweler store. Every item i^{th} in the store is associated with weight $W_i$and value Vi. Both wi and vi are positive integers. Maximum capacity of his knapsack is M where M>0. Thief decided to pick as much load as possible subjected to constraint that total weight of load should not exceed M[3][4]. If whole of the item can't be accommodated in knapsack then fraction of it can be enclosed in knapsack. Before applying greedy strategy to find subset of items that can generate an optimal load, we need to scrutinize whether problem has two key elements of greedy heuristic or not.

ISSN: 0971-1260
Vol-22-Issue-3-July-2019

i. Greedy-choice property
ii. Optimal substructure

Let $I = \{I1, I2, \ldots . Ip\}$set of p items, $V = \{V1, V2, \ldots . . Vp\}$ and $W = \{W1, W2, \ldots . . Wp\}$are set of values and weights associated with each item p respectively. Capacity of knapsack is M.
S is an optimal solution that consists of subset of n items such that total weight of items in subset is equal to M and value of knapsack is maximum[5]. Assume k is one of the item from S, if we discard some weight m of k from S, then it gives another equivalent optimal solution S' where persisting weight of (n-1) items in addition to (W_k– m) weight of item k will be equal to M-m and value of knapsack will be still maximum. Hence, we can say that fractional knapsack problem is an optimal substructure.
To solve the problem, Vi/Wiratio is computed for every i^{th}item. It is termed as value density. Item are sorted in the decreasing order of value density[6]. First item from the list is added in the knapsack; if item is completely exhausted and knapsack is still left with some free space then maximum possible fraction of second item is put inside knapsack. Procedure is repeated till the maximum capacity of knapsack is reached.

1.2.Huffman Code

Huffman code tries to compress the data by assigning variable length binary code to each character[7]. Every character is associated with frequency of its occurrence and length of its code depends on how frequent it appears[8]. The most frequently occurring character has smallest length of code and least frequently used character has largest code length.

	a	B	c	d	e	f
Frequency	40	35	12	13	20	10
Binary Code	0	10	1100	1110	1111	1101

ISSN: 0971-1260
Vol-22-Issue-3-July-2019

Table1. A data file comprises of 5 characters a,b,c,d,e, and f. Every character has frequency that specifies how many times character is popping up in the file. Huffman code assigned binary string to every character.

The mapping of characters and binary code should be done in such a manner that it does not lead to ambiguity. To isolate one code from another, code of any character is restricted to be prefix of rest of the characters. Codes that echo this property are **Prefix Code.**

Huffman's greedy algorithm

Huffman composes optimal prefix code by making greedy choices to avoid collision. Let A represents set of m characters[9]. Every character "a" belonging to A has an attribute a.freq that determines its frequency. Greedy algorithm will build a tree T using bottom approach. It will start with |A| leaf nodes and apply |A| - 1 merging operations. Two leaf nodes with minimum frequency are merged to create a new internal node that stores their sum. The algorithm is implemented using **MIN-PRIORITY** queue that takes freq as a key to choose least frequent characters[6]. Following is the pseudo code for making greedy choices.

HUFFMAN(A)

1. m=|A|
2. Q=A
3. **For** j=1 **to** m-1
4. Create a new internal node x
5. x.left=FETCH-MIN(Q) // assume y is selected
6. x.right=FETCH-MIN(Q) // assume z is selected.
7. x.freq=y.freq+z.freq
8. Insert(Q,x)
9. **Return** FETCH-MIN(Q)

1.3.Matroid

Hassler whitney invented matroid in 1935. A problem can be optimized by greedy approach if it is proved to be a matroid. A matroid S=(M,I) should satisfy following axioms[10][11]:

i. M is a finite set.

ii. I is a non-empty **independent** subsets of M and should satisfy **hereditary** property[12]. Property states that if $X \varepsilon I$ and $Y \subseteq X$ then $X \varepsilon I$.

ISSN: 0971-1260
Vol-22-Issue-3-July-2019

Let G is an undirected graph such that G=(V,E) where V={a, b, c, d, e, f} is a finite set of vertices and E = {{a},{b},{c},{d},{e},{f},{a,b},{b,c},{c,d},{d,e},{e,f},{a,b,c,d,f},{a,b,c,e},{a,b,c,d,e,f}.....} is a non- empty independent set of edges that are subsets of E. Assume, X={ a,b,c,d,f} such that $X \varepsilon E$. And $Y = \{a, b, c\}$ be subset of X.Since Y also belongs to E, hence E satisfies hereditary property

iii. S should satisfy Exchange property. It states that, if$P \varepsilon I$, $Q \varepsilon I$ and |P| < |Q|, then ∃ some element $Y \varepsilon\ Q\ -\ P$ such that $P \cup Y\ \varepsilon\ I$.

Let, $P = \{e, f\}, Q = \{a, b, c, d, f\}\ s.t\ P, Q\ \varepsilon\ I$ from graph G

Where $|P|\ = 2\ , |Q| =\ 5\ \therefore\ |P| < |Q|$

$\{a, b, c, d, f\}\ -\ \{e, f\}\ =\ \{a, b, c, d\}$ (By computing Q-P)

$$Then\ \exists\ some\ element\ \ Y\ \ =\ \{c\} \varepsilon\ Q\ -\ P, such\ that\ Y \cup P\ \varepsilon\ E.$$

1.3.1. Task scheduling problem as a matroid

Problem focuses at scheduling n number of jobs of unit time on single processor such that tasks are completed before their deadline. If task is not completed within the deadline then it incurs penalty[6].

INPUT:

- $T = \{t1, t2, t3, \ldots . tn\}$Set of n tasks of unit time.
- $. D = \{d1, d2, d3, \ldots . dn\}$ Set of deadlines such that 0din for i=1,2,3...n
- $P = \{p1, p2, p3, \ldots . pn\}$Set of penalties. Task$t_i$is incurred with penalty $p_i$if it's not completed within deadline di.

OUTPUT:

- An optimized schedule S that has incurred minimum penalty.

Set A is said to be an independent set of tasks if a schedule where every task finishes before its deadline. $Nt(A)$ signifies count of task in set A that have deadline before or equal to t. For t=0, count will be zero.

PROOF:

Consider two independent sets of tasks X and Y and $|X| > |Y|$. Assume m is the largest t s.t.$Nt(X) \leq Nt(Y)$. As $Nn(X)\ = |X|$ and $Nn(Y) = |Y| but\ |X| > |Y|$, that means m < n and $Nj(X)\ > Nj(X) for\ m + 1 \leq j \leq$n. ∴X has more tasks with deadline m+1 and more than set Y. Suppose $bi\ \varepsilon\ X - Y$with deadline $m + 1\ s.t\ y' = ybi$. Since Y is independent, then $Nt(y') =$

ISSN: 0971-1260
Vol-22-Issue-3-July-2019

$Nt(Y)t$ for 0tm. Since X is independent, then $Nt(y') \leq Nt \leq (X)t\ for\ m < t \leq n$.∴y'is independent, hence (A,I) is a Metroid.

2. Conclusion

Greedy algorithm works on sub-solutions with the intention to solve entire problem. In this paper, we have discussed fractional knapsack problem, Huffman code and Metroid. It has been proved that the above three problems can be solved by greedy approach as they satisfy global sub-structure and greedy choice property.

Bibliography

[1] A. S. M. V. S. Annu Malik, "Greedy Algorithm," International Journal of Scientific and Research Publications, vol. 3, no. 8, 2013.

[2] lJérémieDubois-Lacoste, F. Pagnozzi and T. Stützle, "An iterated greedy algorithm with optimization of partial solutions for the makespan permutation flowshop problem," Computers & Operations Research, vol. 81, pp. 160-166, 2017.

[3] S. L. Kanagala, "A Study of Analyzing Greedy Approach for fractional knapsack problem," International Journal of Advanced Research in Computer and Communication Engineering, vol. 5, no. 2, 2016.

[4] M. S. Veenu Yadav, "A Review Paper on Solving 0-1 knapsack Problem," International Journal of Computer Science and Information Technologies, vol. 7, no. 2, 2016.

[5] P. S. .K, M. T. .V. and C. .Suhas, "A Study of Performance Analysis on Knapsack Problem," International Journal of Computer Applications, 2016.

[6] L. L. B. Korte, "Mathematical structures underlying greedy algorithms," in Fundamentals of Computation Theory - Proceedings of the 1981 International FCT-Conference, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).

[7] A. Malik, N. Goyat and P. V. Saroha, "Greedy Algorithm: Huffman Algorithm," International Journal of Advanced Research in computer science and software engineering, vol. 3, no. 7, 2013.

[8] F. Codeword, Greedy Algorithms: Huffman Coding, cs.umd.edu.

[9] S. B. L. E. A. A. FM Rahman, Study on Selected Greedy Algorithms, dspace.uiu.ac.bd, 2018.

[10] S. A. Haider, "Matroids And Greedy Algorithms A Deeper Justification of Using Greedy Approach To Find A," Annales UMCS, Informatica, 2019.

[11] D. L. Neel and N. A. Neudauer, "Matroids You Have Known," Mathematics Magazine, vol. 82, no. 1, pp. 26-41.

[12] B. Korte and L. Lovasz, "Greedoids - A structural framework for the greedy algorithm," in Progress in Combinatorial Optimization, Academic process, pp. 221-243.