International Journal of Technical Innovation in Modern Engineering & Science (IJTIMES)
Impact Factor: 5.22 (SJIF-2017), e-ISSN: 2455-2585
Volume 4, Issue 7, July-2018

# Approximation Algorithms (A Review)

Shakeel Ahmad Dar

J&K State Board of Technical Education

***Abstract: Solutions to Problems of mathematics are transformed into Algorithms and how easily, in terms of space and time, the algorithm solves the problem determines the efficiency of the algorithm .Certain problems of mathematics find various types of algorithms for their solution while there are certain categories of problems where in it is difficult to find the algorithm that solves it and sometimes even the algorithm found is very inefficient. This paper reviews a set of algorithms whose solution is not in polynomial time but can be approximated in polynomial time.***

***Keywords: NP class; Algorithms; Vertex Cover; Polynomial time;LPT;***

**Introduction:**

Finding the best possible solution from all the feasible solutions for a problem forms the crux in optimization problems. This best possible solution is to find the minimum value or maximum value (based upon the type of problem) for a variable or many variables. The Problems that can be solved by a deterministic Turing machine in Polynomial time are called as **P** Problems. However there is a superset of these P Class problems which can be solved in Polynomial time but only by a Non-Deterministic Turing Machine. This set of Problems is referred as NP Class.

There is another set of Problems referred as NP-Complete and NP-hard. A Problem is said to be NP-Hard if every problem in NP class is reducible to this problem in Polynomial time. On another hand a problem is said to be NP-Complete if it is NP-Hard besides being in NP Class. The following diagram illustrates the relation between P, NP, NP-Hard and NP-Complete with the assumption that NP != P.

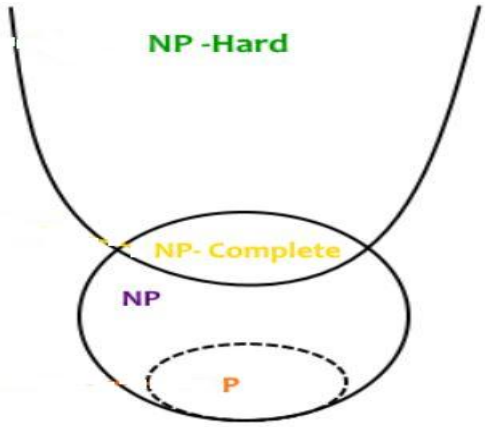


**Fig 1: Relation between P, NP, NP-Complete and NP-Hard.**

**Tackling NP-Hard Problems:**

Since P!=NP (at least as of writing this paper), it is not exactly possible to find the optimal solution of a NP hard Problem in Polynomial time. The strategy for tackling NP-Hard Problems can be :

*a) Brute-Force Techniques:* Brute Force Technique guarantees the finding of optimal solution as clever enumeration techniques are developed. But there is no guarantee of time . It is not possible to find the optimal solution to all NP hard Problems on every instance of input.

*b) Heuristic Techniques:* Development of some intuitive algorithms which are guaranteed to run in polynomial time but there is no guarantee on quality of solutions.

However there is a middle path between the above two techniques to tackle NP-hard Problems, the algorithms commonly referred as Approximation Algorithms which guarantee the time complexity of the solution to be in Polynomial time. Although the algorithm does not necessarily get an optimal solution but it guarantees a solution which is very close to the optimal solution.

**Approximation Algorithms:**

Definition*:* Given an optimization problem P, an algorithm A is said to be an approximation algorithm for P, if for any given instance I, it returns an approximate solution that is a feasible solution.

Terminology*:*

*P: Represents an NP-Hard Problem such as 0/1 Knapsack or Travelling Salesman Problem or Vertex Cover Problem etc.*

*I: Represents an Instance of the problem P.*

*F*(I): Represents an Optimal Solution of the Problem P to Instance I.*

*F^(I):Represents feasible solution of the Problem to Instance I by the approximation algorithm.*

*A: Represents an algorithm that generates a feasible solution to every instance I of problem P.*

For Maximization Problems, F*(I)>F^(I),

For Minimization Problems, F*(I)<F^(I)

Depending upon the difference between Optimal Solution and feasible solution found using the approximation algorithms, different categories of approximation schemes have been established. They are

1. **Absolute Approximation**: A is absolute approximation algorithm for problem P if and only if for every instance I of P,

   **| F*(I)-F^(I) | <=K** for some constant K.

   Planar Graph Coloring is an example of absolute approximation algorithm.

Brief Illustration: To determine the minimum number of colours needed to color a planar graph. It is already proved that all planar graphs are 4-colorable. If the number of vertices in a graph is Null, then it is 0-Colorable, if the number of edges in the graph is null, then it is 1-colorable and the graph is 2-Colorable if the graph is bipartite. In other cases determining the graph is 3-colorable is actually NP-Hard Problem. So if an algorithm is designed which checks the null vertex case, null edge case and bi-partite case and returns 0,1 or 2 and in all other cases return 4. Thus for such kind of algorithms, the complexities of Optimal and Feasible solution can be expressed as

**| F*(I)-F^(I) | <=1.**

2. **f(n)-Approximation (Polynomial Time Approximations) :** A is an f(n)-approximation for the problem P if and only if for every instance I of size n,

$$\frac{|F^*(I)-F^(I)|}{F^*(I)} <= f(n)$$

**Vertex Cover Problem:**

Vertex Cover Problem falls in **f(n)-Approximation (Polynomial Time Approximations)** category of Approximation Algorithms. A vertex cover of an undirected graph is a subset of its vertices such that for every edge (u, v) of the graph, either ‘u’ or ‘v’ is in vertex cover. Although the name is Vertex Cover, the set covers all edges of the given graph. Given an undirected graph, the vertex cover problem is to find minimum size vertex cover. Fig-2 shows some examples of graphs where min vertex cover is of 0 ,1 and 2 .

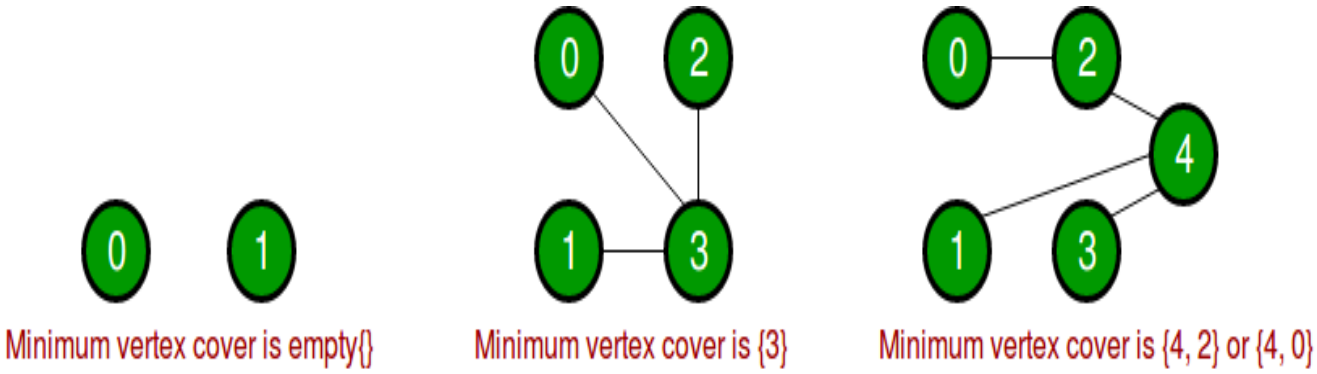


**Fig-2: Vertex Cover Problems**

Vertex Cover Problem is a known NP Complete problem, i.e., there is no polynomial time solution for this unless P = NP. There are approximate polynomial time algorithms to solve the problem though.

One of the approximate algorithms to solve the problem is as

*FindMinVertex (Graph (V, E))*

*{*

*1) X= {}*

*2) While (E is Not Empty) a) Pick any arbitrary edge (u, v) from E.*

*b) Add u, v in X.*

*c) Remove all edges from E with u or v as their vertices.*

*3) Return X.*

*}*

The Execution of the above algorithm is shown as in Fig 3.

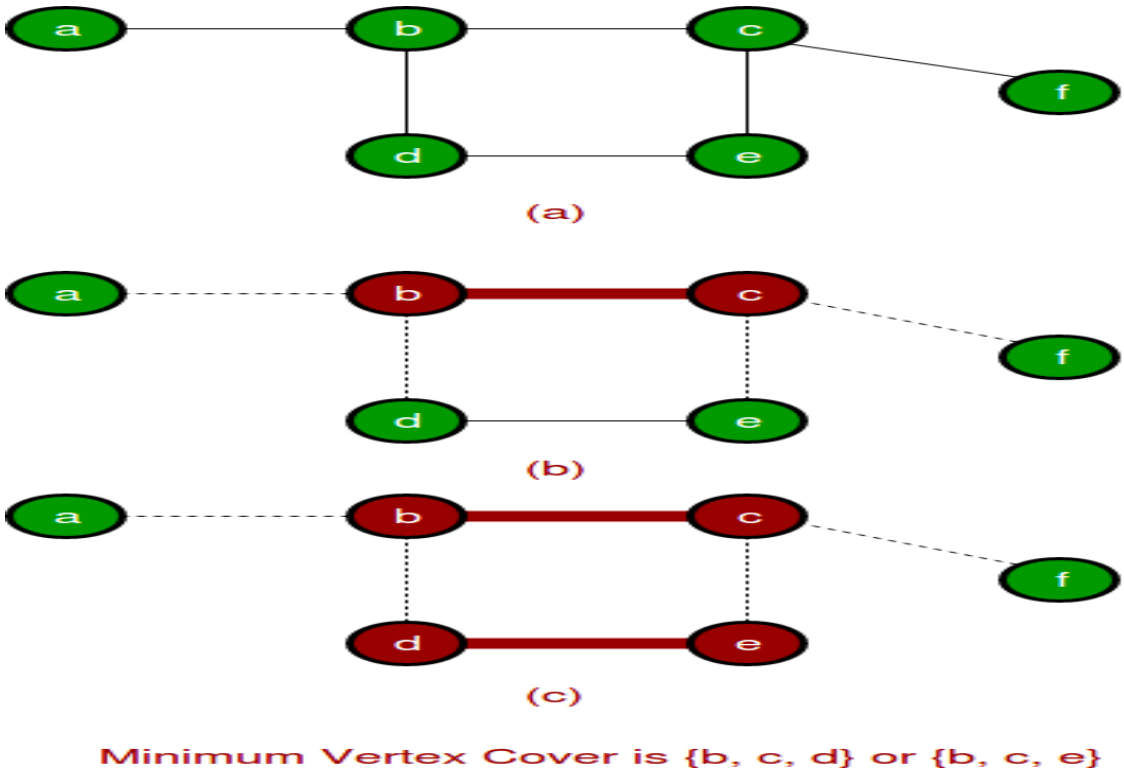


**Fig-3: Finding Minimum Vertex Cover for a Graph using Approximation Algorithm**

The running time of this algorithm is O(V+E) when the graph is represented as Adjacency List. The above Vertex Cover Algorithm is guaranteed to yield a vertex cover which is no more than twice the optimal vertex cover of the graph. To Prove it lets assume that A be the set of edges that are chosen arbitrarily. The Optimal Cover must include at least one end point of each edge, thus

|F*(I)|>=|A|

Also no two edges in A share End-Point, since an edge is picked and all edges incident on its end points are removed from the edge List. Thus

|F^(I)=2|A|.

Thus

|F*(I)|/|F^(I)|>=2

OR

**F*(I)<=2|F*(I)|.**

3. **€- Approximation:** A is an €-Approximation algorithm for problem P iff A is f(n)-Approximation for which f(n)<= € for some small constant €.

An example of €-Approximation is **Scheduling Independent Task problem on a Symmetric Multiprocessor Systems (With number of Processors>2)**

The Problem of finding the minimum finish time schedule on m-identical Processors for a set of tasks n, is NP-Hard.

However there exists a very simple scheduling rule that gives a very near solution to the optimal solution and that approximation falls in **€-Approximation Category.** The Scheduling rule is known as the **Largest Processing Time (LPT) rule** and the schedule obtained from this rule is called as LPT Schedule and as per the rule , whenever any processor becomes free , it is assigned the task among all the unassigned task with the max processing time . In case 2 or more tasks are having same processing time, arbitrary decision is taken.

Example:

**Case 1:**

Let m(number of Processors)=3, n(number of tasks)=6

Processing times of 6 tasks are as

(T1, T2, T3, T4, T5, T6)= (8, 7, 6, 5, 4, 3).

The LPT Schedule is given as Table 1.

The finish time of this schedule is 11.

**Table 1: LPT Schedule of (T1,T2,T3,T4,T5,T6)=(8,7,6,5,4,3) with Finish Time 11.**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| P1 | T1 | | | | | | | | T6 | | |
| P2 | T2 | | | | | | | T5 | | | |
| P3 | T3 | | | | | | T4 | | | | |

In above Case-I, the Total Processing Time of all the tasks is 8+7+6+5+4+3=33 and the optimal solution would give the minimum time as 11 for 3 processors when they are distributed among them. Thus in this case the LPT Schedule is in fact the optimal solution.

**Case 2:**

Let m=3, n=7 with the processing times of the tasks as

(T1, T2, T3, T4, T5, T6, T7)=(5, 5, 4, 4, 3, 3, 3)

The LPT Schedule of this case is as shown in Table 2, with finish time as 11.

**Table 2: LPT Schedule of Case 2 with finish time 11.**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| P1 | T1 | | | | | T5 | | | T7 | | |
| P2 | T2 | | | | | T6 | | | Unused | | |
| P3 | T3 | | | | T4 | | | | | | |

However the Optimal solution of this case yields 9 as finish time as shown in table 3.

**Table 3: Optimal Schedule of Case 2 with finish time 9.**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| P1 | T1 | | | | | T4 | | | |
| P2 | T2 | | | | | T3 | | | |
| P3 | T5 | | | T6 | | | T7 | | |

It has been established that the LPT problem basically is sorting problem and hence can be solved in O(nlog(n)) time and as per Graham's Theorem

$$\frac{|F*(I)-F\hat{}(I)|}{F*(I)} <= \frac{1}{3} - \frac{1}{3m}$$ for m number of processors.

**Conclusion:** In this paper I have elaborated on the general concepts of Approximation Algorithms and compared certain examples with the solutions obtained though approximation algorithms and the actual optimal solutions wherein the difference exists is substantial but it is far lower than the cost that will be incurred to find the optimal solution. More work needs to be done in the field of Approximation Algorithms which should aim to further decrease the difference between the optimal and approximate solutions.

**References:**

**1.** Arora, S. & Lund, C. (1997) in Approximation Algorithms for NP-Hard Problems, ed. Hochbaum, D. S. (PWS, Boston), pp. 399–446.

2. Barahona, F., Gro¨tschel, M., Ju¨nger, M. & Reinelt, G. (1988) Oper. Res. 36, 493–513.

3. Motwani, R. & Raghavan, P. (1995) Randomized Algorithms (Cambridge Univ. Press, New York).

4. Goemans, M. X. (1997) Math. Prog. 79, 143–161.

5. Hochbaum, D. S., ed. (1997) Approximation Algorithms for NP-Hard Problems (PWS, Boston).