Design and Analysis of Algorithms

(Prof. Parminder Kaur Wadhwa)

Date

Page

Approximation Algorithms

The best-known algorithms for NP-hard problems have a worst-case complexity that is exponential in the number of inputs.

There is still plenty of room for improvement in an exponential algorithm even if we try to make them NP-complete.

Many NP-hard optimization problems have great practical importance and it is desirable to solve large instances of these problems in a reasonable amount of time.

If we are to produce an algorithm of low polynomial complexity to solve an NP-hard optimization problem, then it is necessary to relax the meaning of "solve".

There are two relaxations of the meaning of "solve":-

(1) In the first we remove the requirement that the algorithm that solves the optimization problem P must always generate an optimal solution.

This requirement is replaced by the requirement that the algorithm

for P must always generate a feasible solution with value close to the value of an optimal solution.

A feasible solution with value close to the value of an optimal solution is called an approximate solution.

An approximation algorithm for P is an algorithm that generates approximate solutions for P.

In the case of NP-hard problems, the exact solutions (i.e. optimal solutions) may not be obtainable in a feasible amount of computing time.

But, one can get an approximate solution using a reasonable amount of computing time.

(2) In the second relaxation, we look for an algorithm for P that almost always generates optimal solution.

Algorithms with this property are called probabilistically good algorithms.

Approximation Algorithms (in detail)

Let P ⇒ be a problem
I ⇒ be an Instance of problem P

$F^*(I)$ ⇒ be the value of an optimal solution to I

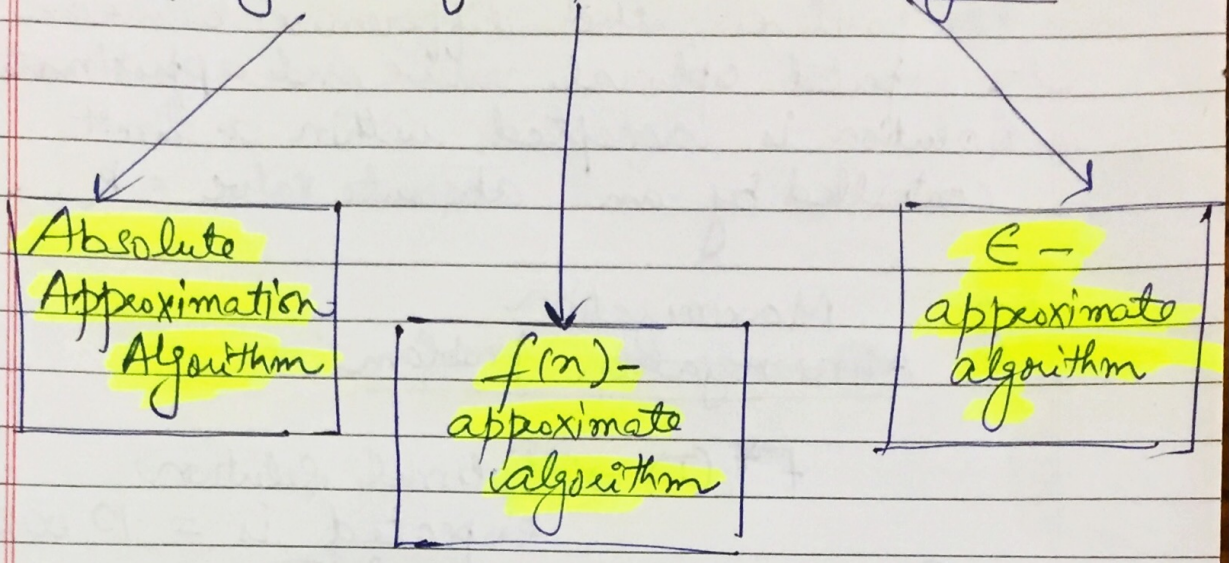
$\hat{F}(I)$ ⇒ be the value of the feasible solution generated by A.

An approximation algorithm generally produces a feasible solution to I whose value $\hat{F}(I)$; → is as follows:-

$\hat{F}(I) < F^*(I)$ if P is a maximization problem

$\hat{F}(I) > F^*(I)$ if P is a minimization problem

Categories of Approximation Algorithms



1. Absolute approximation algorithm :-

A is an absolute approximation algorithm for problem P if and only if for every instance I of P :-

$$|F^*(I) - F\hat{}(I)| \leq k$$

for some constant k.

The variation or difference between expected optimal solution and the approximate solution is within a limit k, (less than or equal to k), (k is set in the algorithm).

The difference will be negative or zero in case of minimization problem and positive or zero for maximization problem.

Either negative or positive is the difference, their absolute value is checked.

The difference between the expected optimal value and approximate solution is accepted within a limit controlled by an absolute value = k.

For example:-

Minimization problem:-

$F^*(I)$ ⇒ expected optimal solution (e.g. min cost) = 10,000

$F\hat{}(I)$ ⇒ computed approximate solution (a feasible solution) is little bit higher than the expected min cost ⇒ e.g. 10,080

Let the absolute value difference acceptable for the problem = k = 100

$\therefore$ $F^*(I) - F\hat{}(I)$

$= |10000 - 10,080| = |-80| = 80$

As, $80 \leq k$ i.e.

$80 \leq 100$, so it (i.e. $F\hat{}(I)$) is accepted as an approximate solution.

e.g. For maximization problem:-

$F^*(I)$ ⇒ expected max profit = 10,000

$F\hat{}(I)$ ⇒ computed solution = 9,950
(a little bit less than expected max profit)

k is set as 80. (for example)

$$|F^*(I) - F\hat{}(I)|$$

$$= |10000 - 9950|$$

$$= |50|$$

$$= 50$$

As, $50 \leq 80$ $(50 \leq k)$
$k = 80$ here!

∴ $F\hat{}(I)$ is accepted as an approximate solution by the absolute-approximate algorithm.

2. $f(n)$ - approximate algorithm

A is an $f(n)$ - approximate algorithm if and only if for every instance I of size n,

$$\frac{|F^*(I) - F\hat{}(I)|}{F^*(I)} \leq f(n)$$

where $F^*(I) > 0$ [denominator is greater than 0]

The absolute difference per optimal (expected) value is computed.

The absolute difference divided by optimal (expected) solution should be within the limit i.e. less than or equal to a function $f(n)$, where n is the size of Instance I.

(3) ϵ - Approximate algorithm

An ϵ-approximate algorithm is an $f(n)$-approximate algorithm for which $f(n) \leq \epsilon$ for some constant ϵ.

Note:-

For maximization problems;

$$\frac{|F^*(I) - F\hat{}(I)|}{F^*(I)} \leq 1$$

for every feasible solution to I.

Hence, for maximization problems we normally require $\boxed{\epsilon < 1}$ for an algorithm to be judged ϵ-approximate.

For example :-

Scheduling Independent Tasks

Obtaining minimum finish time schedules on m, identical processor (where $m \geq 2$) is NP-hard.

An Instance I of the scheduling problem is defined by a set of n task times t_i,

$1 \leq i \leq n$

$n \rightarrow$ no. of tasks
$t_i \rightarrow$ task time

m = number of processors.

The scheduling rule :-

LPT (Longest Processing Time) Rule :-

An LPT schedule is one that is the result of an algorithm that, whenever a processor becomes free, assigns to that processor a task whose time is the largest of those tasks not yet assigned.
(Ties are broken in an arbitrary manner).

Let,

$F^*(I)$ be the finish time of an optimal m-processor schedule for Instance I of the task scheduling problem.

Let $F\hat{}(I)$ be the finish time of an LPT schedule for the same instance,

Then,

$$\frac{|F^*(I) - F\hat{}(I)|}{F^*(I)} \leq \frac{1}{3} - \frac{1}{3m}$$

Let I be an m-processor instance of the scheduling problem.

Let $F^*(I)$ be the finish time of an optimal schedule for I, and

Let $F^\wedge(I)$ be the length of the schedule generated by the following scheduling rule:-

Let k be some specified and fixed integer.

Obtain an optimal schedule for the k longest tasks. Schedule the remaining (n-k) tasks using the LPT rule.

Then,

$$\frac{|F^*(I) - F^\wedge(I)|}{F^*(I)} \leq \frac{1 - 1/m}{1 + \lfloor k/m \rfloor}$$

Solving for k, we obtain that any integer k, $k > \frac{(m-1)}{\epsilon - m}$ guarantees ϵ-approximate schedules.