Approximation Algorithms for NP-problems

Narendhar Maaroju, K.V.R Kumar ,Dr.Deepak Garg

Abstract

Algorithms are at the heart of problem solving in scientific computing and computer science. Unfortunately many of the combinatorial problems that arise in a computational context are NP-hard, so that optimal solutions are unlikely to be found in polynomial time. How can we cope with this intractability? One approach is to design algorithms that find approximate solutions guaranteed to be within some factor of the quality of the optimal solution. More recently, in large-scale scientific computing, even polynomial time algorithms that find exact solutions are deemed too expensive to be practical, and one needs faster (nearly linear time) approximation algorithms. We will consider the design of approximation algorithms for various graph-theoretical and combinatorial problems that commonly arise in scientific computing and computational biology. These include set covers (vertex covers in hypergraphs), matching, coloring, and multiple sequence alignments in computational biology.

Keywords

Approximation, Optimization, MCMC, Online Computation,Randomization.

1 Introduction

Given a problem we first need to determine its complexity status. This is done either by designing a polynomial-time algorithm for its solution or by proving that the problem is ***NP***-hard.

NP draws the class of problems solved by non-deterministic Turing-machines in polynomial time. This class is the usual framework for most of people working in the intersection of two major computer science areas: *Complexity theory and Combinatorial optimization*. The main sub-classes of NP are:

The class P of problems solved in polynomial time by a deterministic Turing-machine; this is the class of the easiest NP problems (commonly called polynomial problems);

[1]

The class NP-complete of the hardest NP problems; for this class the major property is the following: solution of just one of its problems in polynomial time by a deterministic Turing-machine would imply solution of any other NP-complete problem by such machines.

In other words, we do not know if problems in NP\P can be polynomially solved or not.

Fundamental conjecture in complexity theory is that P≠NP, i.e., that NP-complete problems are not polynomial. This conjecture, shared by the whole of the complexity-theory community, entails consideration of NP-world as shown in Figure 1.

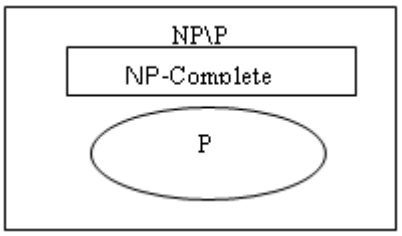

Fig.1 The NP-world as believed by researchers.

Confirmation or infirmation of conjecture P≠NP is the major open problem in computer science and one of the most famous contemporary mathematical problems.

The existence of complete problems for NP has been proved in [4]. The first such problem is SAT, where one asks for the existence of a model for a first-order propositional formula described in conjunctive normal form.

Once we know that our problem is *NP*-hard, we need to determine whether we want its exact solution or may be happy with an approximate solution. It is unlikely that an exact solution can be found by a polynomial-time algorithm. An exact solution can be found by various methods of reduced enumeration, typically by a branch-and-bound algorithm. Alternatively, the problem can be formulated as a mathematical programming problem. In any case, for problems of practical interest only small-size instances can be handled by exact methods.

In order to find a "good" solution within an acceptable amount of time, two types of algorithms can be developed:

- approximation algorithms;

[1] Manuscript received April15, 2009.

K.V.R.Kumar is with the Thapar University, Patiala, India (e-mail:skumar98765@gmail.com).

Narendhar Maaroju is with the Thapar University, Patiala, India (e-mail:maroju.83@gmail.com).

Dr.Deepak Garg is with the Thapar University, Patiala, India (e-mail:deepakgarg@ieee.org).

- heuristic algorithms.

An algorithm is called an ***approximation*** algorithm if it is possible to establish analytically how close the generated solution is to the optimum (either in the worst-case or on average).

The performance of a ***heuristic*** algorithm is usually analysed experimentally, through a number of runs using either generated instances or known benchmark instances. Heuristic algorithms can be very simple but still effective (dispatching rules). Most of modern heuristics are based on various ideas of ***local search*** (neighbourhood search, tabu search, simulated annealing etc.).

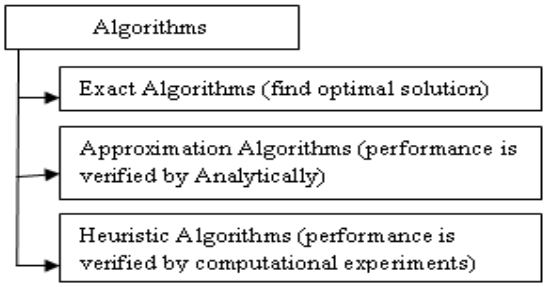

Fig.2 Shows the type of Algorithms

Approximation algorithms produce solutions in ***polynomial time***, but for the price of loss of optimality. The solutions found are ***guaranteed*** to be within a fixed percentage of the actual optimum.

Heuristic algorithms produce feasible solutions, which are not guaranteed to be close to optimum.

"Goodness" of approximation algorithm can be estimated by the ratio

$$\frac{F(S_A)}{F(S_{OPT})}$$

where $F(S_A)$ is the value of the objective function for the solution S_A obtained by the approximation algorithm and $F(S_{OPT})$ is the value of the objective function for the optimal solution.

We define a *r-approximation algorithm* to be an algorithm that runs in polynomial time and delivers a solution of value at most *r* times the optimum for any instance of the problem, i.e.,

$$\frac{F(S_A)}{F(S_{OPT})} \leq r$$

The value of *r* is called a worst-case ratio bound.

2 Approximation and Optimization

Complexity theory deals with decision problems; these problems are defined as questions about *existence* of solutions of a certain value verifying some specific properties, and solution of such problems is a correct *yes* (or *no*) statement. Classes NP and NP-complete are defined with respect to decision problems. On the other hand, optimization deals with *computation* of optimal solutions. Any optimization problem has a decision version. The optimization counterparts of NP, NP-complete and P are NPO, NP-hard and PO respectively. NPO is the class of optimization problems whose decision version is in NP;

- NP-hard is the class of optimization problems having NP-complete decision counterparts;
- PO is the class of polynomial optimization problems. NPO

Formally, an NPO problem Φ is defined as a

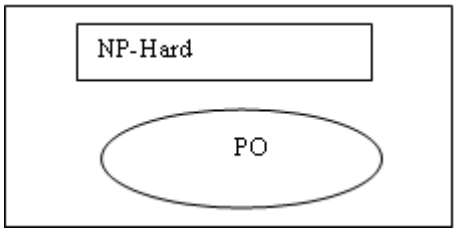

Fig.3 The NP-world as believed by researchers.

quadruplet $(I\Phi, \text{SOL}\Phi, m\Phi, opt\Phi)$, where:

- $I\Phi$ draws the set of instances of Φ;
- $\text{SOL}\Phi(I)$ draws the set of feasible solutions of an instance $I \in I\Phi$;
- $m\Phi(I,S)$ is the measure of a solution $S \in \text{SOL}\Phi(I)$ (the objective value of S);
- $opt\Phi \in \{min, max\}$.

Moreover, any $I \in I\Phi$ and $S \in \text{SOL}\Phi(I)$ are recognizable in time polynomial in $|I|$, and, for any $I \in I\Phi$ and $S \in \text{SOL}\Phi(I)$, $m\Phi(I,S)$ is computable in time polynomial in $|I|$. We will denote by $opt\Phi(I)$ the optimal value (the value of the optimal solution) of I and by $\omega\Phi(I)$ the worst value of I, i.e., the value of the worst feasible solution of I. This solution can be defined [6,9] as the optimal solution of the NPO problem $\Phi' = (I\Phi', \text{SOL}\Phi', m\Phi', opt\Phi')$ where $I\Phi' = I\Phi$, $\text{SOL}\Phi' = \text{SOL}\Phi$

and $opt\Phi' = \max$ if $opt\Phi = min$

and $opt\Phi' = \min$ if $opt\Phi = max$.

Since it is believed as very unlikely that a polynomial time algorithm could ever be devised to exactly solve Φ, the area of *polynomial approximation* is interested in algorithms computing sub-optimal **feasible** solutions of *I* in polynomial time. Such algorithms are called *polynomial time approximation algorithms*.

A given problem can admit several polynomial time approximation algorithms, each one providing a distinct feasible solution.

• *How can we classify these algorithms with respect to their ability in approximating the optimal solution of a generic instance I of the problem considered?"*

• *"How one can decide that an algorithm is better – more adapted -- than another one supposed to approximately solve the same problem?"*

To answer to the above questions, we need measures that count the "goodness" of an approximation algorithm A following some predefined criteria. The

most common such criteria try to answer to one of the following questions (in what follows we consider Φ fixed; so we simplify notations by eliminating it as index):

• **Q1**: by how *m(I,S)* differs from opt*(I)*?

• **Q2**: how many times is *m(I,S)* greater than (smaller than when dealing with maximization problems) opt*(I)*, in other words, what is the relative distance of *m(I,S)* from opt*(I)*?

• **Q3**: how *m(I,S)* is positioned with respect to opt*(I)* and to *ω(I)*, or more precisely, to what percentage of the best feasible values the value computed is lying?

Question **Q1** induces the measure which in literature is called *absolute performance guarantee*; it is defined as

$ra(A,I) = |m(I,S) - opt(I)|$.

Question **Q2** induces the *relative performance guarantee*, or *standard approximation ratio*; it is defined as $r(A,I) = m(I,S)/opt(I)$.

Finally, question **Q3** induces the differential approximation ratio; it is defined as $\delta(A,I) = |\omega(I) - m(I,S)|/|\omega(I) - opt(I)|$.

For more details about these ratios, the interested reader can be referred to ([2, 7, 8, 10, 11]). With respect to any performance guarantee, one can partition the obtained results into three main categories:

Positive results:

Here, one affirms that, for a given NP-hard problem, a given polynomial time approximation algorithm guarantees a performance measure bounded above (below when dealing with maximization problems) by a fixed constant;

Negative results:

Here, one affirms that, for a given NP-hard problem, no polynomial time approximation algorithm can guarantee a certain approximation level, unless an unlikely complexity theory condition (for example= **NP**) holds;

Conditional results:

These results relate: either two types of approximation behaviour of the same problem or the approximation behaviour of a given NP-hard problem to the approximation behaviour of another one.

Results for the two last categories are, in fact, reductions preserving certain types of approximability properties of the problems involved. Polynomial approximation has two main issues:

- *operational*
- *structural.*

What is at operational stake, is the fast and good solution of natural - real-world - problems, solution of which is a sine qua non condition in order that complex human systems can efficiently function.

Therefore, such a way to apprehend the problem at hand is impossible. On the other hand, its apprehension by heuristics that do not guarantee always feasible solutions (and, furthermore, are probably not so fast) can be also unfeasible, even economically, or socially, or … dangerous. The use of polynomial time approximation for such problems, even if it does not guarantee computation of the best solution, it firstly guarantees solutions respecting all the constraints of the problem dealt and, moreover, these solutions are as near-optimal as possible.

As in classical theory of algorithms, we can extend the classification of approximation algorithms to the classification of the problems solved by them. To be more precise, let us focus ourselves to the standard approximation ratio, denoted above by *r*, and consider a classification of approximFation algorithms following the value of this ratio. Any such value induces what is commonly called *approximation level*.

The most popular approximation levels are the following:

Constant ratio: An algorithm is called constant-ratio algorithm when its approximation ratio *r* is a fixed constant, i.e., when its ratio does not depend on any parameter of the instance *I*;

Polynomial time approximation schema: Here we have a sequence of polynomial (in $|I|$) time algorithms receiving as inputs the instance *I* of an NP-hard problem and a fixed constant $\varepsilon > 0$; for any such ϵ, they compute a feasible solution guaranteeing approximation ratio $1+\epsilon$ ($1-\epsilon$ when dealing with maximization problems);

Fully polynomial time approximation schema:

It is a polynomial time approximation schema but the complexity of any member of the sequence is polynomial both in $|I|$ and $1/\epsilon$;

Logarithmic ratio: when the approximation ratio of the algorithm is $O(\log|I|)$;

Polynomial ratio: when $\exists\ \epsilon>0$ such that the approximation ratio achieved is $O(n^{1-\varepsilon})$ (or when dealing with maximization problems).

Classification of algorithms can be extended to classification of problems considering as approximation level of a problem the level of the best (known) approximation algorithm solving the problem under consideration. We so have the following most popular approximability classes:

APX: the class of NP-hard problems solved by constant-ratio polynomial time approximation algorithms;

PTAS: the class of NP-hard problems solved by polynomial time approximation schemata;

FPTAS: the class of NP-hard problems solved by fully polynomial time approximation schemata;

Log-APX: the class of NP-hard problems solved by polynomial time approximation algorithms achieving ratios logarithmic in $|I|$;

Poly-APX: the class of NP-hard problems solved by polynomial time approximation algorithms achieving ratios polynomial in $|I|$.

Relationships between complexity and approximation are even deeper since one can also prove completeness for some approximation classes. Recall that given a complexity class **C** and a reduction R, a problem Φ is **C**complete under reduction R if $\Phi \in$ **C**

and for any other problem Φ′∈**C**, Φ′ R-reduces to Φ. Considering a complexity class, a complete problem for it can be thought as a hardest problem for the class under consideration. An easy way to think about

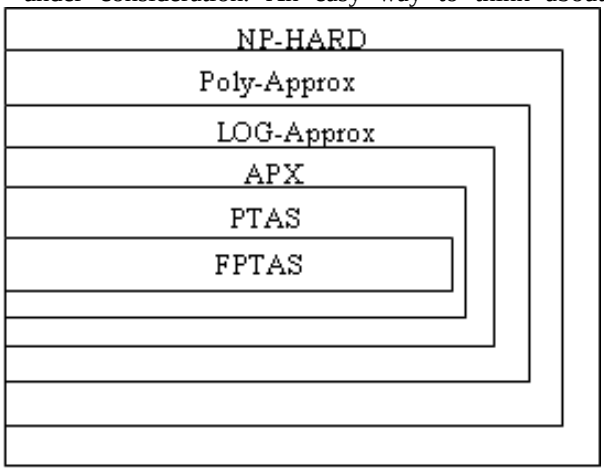

Fig 4. The main approximability classes (under the hypothesis P≠NP) for standard approximation.

completeness of Φ is to say that Φ is in **C** but not in a "superior" class. For example, Φ is NP-complete means that it belongs to NP but not to P (unless P= NP). This can be extended for approximability classes. For an approximability class **C** (recall that such a class, e.g. **APX,** is the set of problems approximable within approximation ratios the value of which correspond to a certain level, e.g. fixed constant ratios) a problem is **C**-complete (e.g., **APX**-complete) if it belongs to **C** (e.g., **APX**) but it cannot belong to a "superior" approximability class (e.g., **PTAS**) unless an unlikely complexity condition holds (e.g., **P**= **NP**).

Note**,** finally that the class of problems solved by fully polynomial time approximation schemata (**FPTAS** when dealing with standard approximation), considered as the highest approximability class, cannot have complete problems since the only higher approximation level is class **PO** (that can be considered as the class of problems polynomially solved within approximation ratio 1 [1,3,5]).

3 Different Approaches for Approximation

3.1 Randomized Approximation Algorithms

A **randomized algorithm** or **probabilistic algorithm** is an algorithm which employs a degree of randomness as part of its logic. The Randomized algorithm typically uses uniformly random bits as an auxiliary input to guide its behavior, in the hope of achieving good performance in the "average case" over all possible choices of random bits. Formally, the algorithm's performance will be a random variable determined by the random bits; thus either the running time, or the output (or both) are random variables.

If we have a maximization problem we say that algorithm A is a factor δ approximation algorithm if on everyproblem instance I with optimal solution value OPT(I),

Let $E[X]$ denote the expectation of the random variable X. For a randomized algorithm, A, we say that A is a factor δ randomized approximation algorithm if on every problem instance I with optimal solution value $OPT(I)$,

$$E[A(I)] \geq \delta . OPT(I)$$

Where $A(I)$ is the value of the solution produced by A on instance I, the expectation is taken over the random choices made by the algorithm, and $\delta \leq 1$.

Simplicity is the first and foremost reason for using randomized algorithms. There are numerous examples where an easy randomized algorithm can match (or even surpass) the performance of a deterministic algorithm.

Speed, for some problems, the best known randomized algorithms are faster than the best known deterministic algorithms.

Lot of witnesses The reason why randomized algorithms are successful in finding the solution is that often the problem instance has a lot of witnesses for "yes" or "no".

Load balancing Randomization is a good way of distributing the load.

Sampling another useful technique is random sampling: pick a small sample out of the large input problem. Solve the problem on the sample and then take back the solution to the original problem.

Symmetry breaking for parallel computation, it is important to break the symmetry.

Derandomization Some simple randomized algorithms can be converted to (not-so-simple) deterministic algorithms via derandomization. Many times the best guarantees on performance for deterministic algorithms are achieved via derandomizing simple randomized algorithms.

3.2 Markov Chain Monte Carlo Method

Markov Chain Monte Carlo (MCMC) methods have become an important numerical tool in statistics (particularly in Bayesian statistics) and in many others scientific areas, to approximately sample from complicated densities in high dimensional spaces .These methods usually require various parameters (e.g. proposal scalings) to be tuned appropriately for the algorithm to converge reasonably well.

An MCMC algorithm constructs a Markov chain that has the target distribution, from which we want to sample, as its stationary distribution. This Markov chain can be initialized with any state, being guaranteed to converge to its stationary distribution after much iteration of stochastic transitions between states. After convergence, the states visited by the Markov chain can be used similarly to samples from the target distribution (see [12] for details).

The canonical MCMC algorithm is the Metropolis method [13], in which transitions between states have two parts: a proposal distribution and an acceptance function. Based on the current state, a candidate for the next state is sampled from the proposal distribution. The acceptance function gives the

probability of accepting this proposal. If the proposal is rejected, then the current state is taken as the next state. A variety of acceptance functions guarantee that the stationary distribution of the resulting Markov chain is the target distribution [14]. If we assume that the proposal distribution is symmetric, with the probability of proposing a new state x^* from the current state x being the same as the probability of proposing x from x^*, we can use the Barker acceptance function [15], giving

$$A(x*; x) = \frac{\pi(x*)}{\pi(x*)+\pi(x)}$$

for the acceptance probability, where $\pi(x)$ is the probability of x under the target distribution.

3.3 Online computation

Online algorithms are the algorithms which have to take decisions based on partial knowledge of the input, e.g., when future inputs are not known. In such situations, randomness is useful because it allows us to defeat an adversary who does not see the random decisions that we are making.

In general, traditional optimization techniques assume complete knowledge of all data of a problem instance. However, in reality it is unlikely that all information necessary to define a problem instance is available beforehand. Decisions may have to be made before complete information is available. This observation has motivated the research on *online optimization.* An algorithm is called *online* if it makes a decision (computes a partial solution) whenever a new piece of data requests an action.

In online optimization the input is modelled as a finite *request sequence* $r_1, r_2, r_3 \ldots$ which must be served and which is revealed step by step to an online algorithm. How this is done exactly, depends on the specific *online paradigm.* The two most common models are the *sequence model* and the *time stamp model.*

Let ALG be an online algorithm. In the *sequence model* requests must be served in the order of their occurrence. More precisely, when serving request $\mathbf{r_j}$, the online algorithm ALG does not have any knowledge of requests $\mathbf{r_i}$ with $\mathbf{i > j}$ (or the total number of requests). When request $\mathbf{r_j}$ is presented it must be served by ALG according to the specific rules of the problem. The serving of $\mathbf{r}_j$ incurs a "cost" and the overall goal is to minimize the total service cost. The decision by ALG of how to serve $\mathbf{r_j}$ is irrevocable. Only after $\mathbf{r}_j$ has been served, the next request $\mathbf{r_{j+1}}$ becomes known to ALG.

In the *time stamp model* each request has a *arrival* or *release time* at which it becomes available for service. The release time $\mathbf{t_j > 0}$ is a nonnegative real number and specifies the time at which request $\mathbf{r_j}$ is released (becomes known to an online algorithm). An online algorithm ALG must determine its behaviour at a certain moment **t** in time as a function of all the requests released up to time **t** and of the current time **t**. Again, we are in the situation that an online algorithm ALG is confronted with a finite sequence $\mathbf{r_1, r_2, \ldots}$ of requests which is given in order of non-decreasing release times and the service of each request incurs a cost for ALG.

3.4 primal-dual method

The primal-dual method for approximation algorithms considers a primal integer programming formulation of the problem in question and the dual of a linear programming relaxation of the integer program. As we will see below, relaxing this constraint in appropriate ways leads to provably good algorithms for *NP*-hard problems in combinatorial optimization. The method yields a solution to the primal integer problem that costs no more than α times the value of the feasible dual solution constructed, which implies that the primal solution is within a factor of α of optimal. The value of the dual solution is always within some factor of α of the value of the primal solution, but may from instance to instance be much closer; by comparing the value of the primal and dual solutions generated, we can give a guarantee for the instance which might be better than α.

The performance guarantee of an algorithm using the primal-dual method is thus connected with the *integrality gap* of the integer programming formulation of the problem. The integrality gap of a formulation is the worst-case ratio over all instances of the value of the integer program to the value of the corresponding linear programming relaxation. Since the performance guarantee of an algorithm using the primal-dual method is proven by comparing the value of a primal solution against the value of a feasible dual, its performance guarantee can never be shown to be better than the integrality gap of the formulation used.

Consider a general LP formulation:

PRIMAL is $\min \quad cx$

$$Ax \geq b$$

$$x \geq 0$$

Its DUAL is $\max \quad yb$

$$yA \leq C$$

$$y \geq 0$$

Complementary Slackness Conditions (CS)

PRIMAL: $x_i > 0 \rightarrow \sum_j y_{ij}\, a_{ji} = c_i$

DUAL: $y_j > 0 \rightarrow \sum_i a_{ij}\, xj = b_j$

4 Analysis and Results

This paper on "**Approximation Algorithms for NP-problems**" describes various concepts, arguments and applications related to the approximation algorithms. The following tables illustrate the comparision of different approaches of Approximation algorithms for different NP-problems.

Table 1
Comparison of different Approximation Approaches.

Approaches for Approximation Algorithms				
	Performance	Approximation guarantee	Feasibility	Resources used
Randomization	1. Performance can be achieved by assigning each vertex with equal probability. 2. This gives us a relative performance bound within constant factor 2. 3. Use semidefinite programming to do a lot better. 4. The reliability of a randomized algorithm can be improved, if more random trials are performed.	1. This is a polynomial-time 2-approximation algorithm means that the solution returned by algorithm is at most twice the size of an optimal.	1. This gives different solutions but all solutions near to optimal. 2. Algorithms that are efficient in that their running times are bounded by a polynomial in the size of the input. 3. Algorithms provably good, in that the solution produced by the algorithm is guaranteed to be close to the optimal solution to within a specified, provable tolerance.	1. They first formulate the problem as an integer program. Next, some constraints in this integer program are relaxed in order that the relaxation be efficiently solvable. Finally, randomization is used to restore the relaxed constraints.
Markov Chain Monte Carlo Method	1 Even when density is known we may have difficulty in generating samples from the distribution corresponding to it. 2. For any state of the Markov chain, there is a positive probability of visiting all other states. 3. Very complex models can be analyzed. 4. Length of the searching phase is difficult to identify.	1. The natural ordering on the state space has no specific role. If there exists an ordering with respect to which the chain is stochastically monotone, we can generate chains starting at minimum and maximum and then stop when they meet. 2. MCMC is only one way to approximate the posterior distribution. It is not a modeling approach itself!	1. MCMC is the general procedure of simulating such Markov chains and using them to draw inference about the characteristics of $f(x)$. 2. In the sense that no rigorous guarantees could be given for the quality of the approximate solutions they produced.	1. MCMC can use resources like, let φ be a very large (but finite) set of combinatorial structures, and let π be a probability distribution on φ. This task is to sample an element of φ at random according to the distributionπ.
Online Computation	1.quality of an on-line algorithm is mostly measured by evaluating its worst case performance. 2. The competitive ratio of a specific on-line algorithm is not the answer to this problem. 3. A lower bound on the competitive ratio of every possible on-line algorithm answers the question! 4. Such lower bounds can be achieved by providing a specific set of instances on which no on-line algorithm can perform well.	1. Numerous problems in operating systems, compilers, graphics, robotics. 2. online algorithms is to improve the performance of one or more existing algorithms for a specific NP-hard problem by adapting the algorithms to the sequence of problem instance(s) they are run on.	1. The number of jobs completed within time T, for some fixed deadline T>0. In particular, our online algorithm's guarantees apply if the job can be written as a monotone, submodular function of a set of pairs of the form (upsilon, tau), where tau is the time invested in activity upsilon.	1. the principle underlying this algorithm is often referred to as the ski principle. it says that once the algorithm has incurred enough total cost by executing a number of cheap actions, the algorithm can afford to take more expensive action, which can be amortized against the collection of cheap actions.
Primal-Dual Method	1. Solving by adjusting dual variables until primal ones are feasible. 2. It reduces the degree of infeasibility of the primal one at the same time.	1. The more trials, the better the approximation.	1. No optimal dual solution needed. 2. The final dual solution is used as a lower-bound for the optimum solution value by means of weak duality.	1. The primal-dual method gives rise to min-max relations which have far-reachibg algorithmatic significance.

5. Conclusion

After analysing the different Approximation approaches for some well-known problems, we conclude that, based on the problem characteristics different approaches are efficient for different problems. We can‟t say particular approximation approach is efficient for all NP-problems. Based on problem criteria and characteristics one of the approach is efficient.

6. References

[1] G. Ausiello, C. Bazgan, M. Demange, and V. Th. Paschos. Completeness in differential approximation classes. Cahier du LAMSADE 204, LAMSADE, Université Paris-Dauphine, 2003. Available on *http://www.lamsade.dauphine.fr/cahiers.html*.

[2] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti- Spaccamela, and M. Protasi, "Complexity and approximation.Combinatorial optimization problems and their approximability properties", Springer, Berlin, 1999.

[3] G. Ausiello, P. Crescenzi, and M. Protasi, Approximate solutions of NP optimization problems, *Theoret. Comput. Sci.*, 150:1-55, 1995.

[4] S. A. Cook, The complexity of theorem-proving procedures. In *Proc. STOC'71*, 151-158, 1971.

[5] P. Crescenzi and A. Panconesi, Completeness in approximation classes, *Inform. and Comput*, 93(2):241-262,1991.

[6] M. Demange and V. Th. Paschos, On an approximation measure founded on the links between optimization and polynomial approximation theory, *Theoret. Comput. Sci*.,158:117-141, 1996.

[7] [M. R. Garey and D. S. Johnson, "Computers and intractability. A guide to the theory of NP-completeness", H. Freeman, San Francisco, 1979.

[8] D. S. Hochbaum, editor. "Approximation algorithms for NPhard problems", PWS, Boston, 1997.

[9] Monnot, V. Th. Paschos, and S. Toulouse, Differential approximation results for the traveling salesman problem with distances 1 and 2, *European J. Oper. Res*., 145(3):557--568, 2002

[10] J. Monnot, V. Th. Paschos, and S. Toulouse, "Approximation polynomiale des problèmes NP-difficiles: optima locaux et rapport différentiel", Informatique et Systèmes d'Information, Hermès, Paris, 2003

[11] C. H. Papadimitriou and K. Steiglitz, "Combinatorial optimization: algorithms and complexity", Prentice Hall, New Jersey, 1981.

[12] W.R. Gilks, S. Richardson, and D. J. Spiegelhalter, editors. *Markov Chain Monte Carlo in Practice*.Chapman and Hall, Suffolk, 1996.

[13] W. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[14] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57:97–109, 1970.

[15] A. Barker. Monte Carlo calculations of the radial distribution functions for a proton-electron plasma. *Australian Journal of Physics*, 18:119–133, 1965.

Narendhar Maaroju Author

This author became a member of IEEE in 2008. Pursuing Master of Engineering (2007-09) in Computer science from Thapar University,Patiala,Punjab India. Member of organizing committee of IACC'09 .

K.V.R.kumar Author

This author became a member of IEEE in 2008. Pursuing Master of Engineering (2007-09) in Computer science from Thapar University, Patiala, Punjab, India. Member of organizing committee of IACC'09 .

Dr.Deepak Garg Author

has done his Ph.D. in the area of efficient algorithm design. He is certified on various technologies from Sun, IBM and Brain bench. He is Senior Member of IEEE and is Executive Member of IEEE Delhi Section. He is Life Member of ISTE, CSI, IETE, ISC, British Computer Society and ACM. He is also serving at different levels in various social and spiritual organizations. He has taught various core subjects at graduate and undergraduate levels and is guiding PhD students.

He has 18 publications in various International Journals and conferences. He is President of Creative Computer Society and Coordinator of various Industry collaborations in the University. He has served in various committees and has been a member of Senate of Thapar University. He has 10 years of teaching/research/development experience that includes working in IBM Corporation, USA. He has edited two books and is on the editorial board of an International Journal. He provides consultancy in the area of software development and technical education.