International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Approximation Algorithms for NP-Hard Problems: A Deep Learning-Based Approach

[1]Aditya, Sagar Choudhary,[2] Abhinav Kumar,[3] Abhinav Tyagi

[1,3,4] B.Tech Student, Department of Computer Science and Engineering, Quantum University, Roorkee, India.

[2] Assistant Professor, Department of Computer Science and Engineering, Quantum University, Roorkee, India.

ABSTRACT

Effectively addressing NP-Hard issues remains a big hurdle in both theoretical computer technological know-how and actual-international packages. While conventional approximation algorithms offer answers which might be close to surest with assured performance bounds, they often fall short in terms of adaptability and scalability, specially in records-wealthy environments. With the rise of deep learning, there's increasing interest in utilizing neural networks to create facts-driven heuristics for tackling these complex challenges. This studies provides a combined method that merges deep gaining knowledge of strategies with traditional approximation techniques to resolve NP-Hard troubles like graph coloring, set cover, and other combinatorial optimization responsibilities. By schooling neural networks to both are expecting elements of the solution or guide heuristic seek tactics, our goal is to enhance both the effectiveness and accuracy of approximate outcomes. Experimental opinions on fashionable datasets spotlight the promising role of deep getting to know in improving approximation techniques for computationally hard issues.

Keywords:Approximate Computing, Computationally Hard Problems, Deep Neural Learning, Graph-Based Techniques, Optimization Strategies, Search Heuristics, Artificial Neural Networks

Introduction

In the field of computational principle and set of rules development, step one toward solving a trouble is to analyze its computational complexity. This normally entails both designing a polynomial-time solution or demonstrating that the hassle is NP-hard. The NP elegance (nondeterministic polynomial time) comprises selection troubles that may be validated in polynomial time by way of a nondeterministic Turing gadget [12].

Numerous challenges in combinatorial optimization and theoretical computer technology fall into this elegance. Within NP, an vital subclass is P, which includes issues solvable in polynomial time the usage of a deterministic Turing device. These are often seemed because the "handiest" problems in NP. On the alternative cease lie the NP-whole troubles, considered the maximum hard within NP. A key characteristic of NP-whole issues is if even considered one of them may be solved in polynomial time, then every hassle in NP may be solved in polynomial time, implying P = NP [4, 7].

That NP-complete troubles are not likely to have polynomial-time answers is a widely held notion, stemming from the conjecture that P ≠ NP [4, 7]. Consequently, locating specific answers for huge NP-difficult problem instances is often computationally impractical. This challenge has motivated the improvement of approximation algorithms—techniques aimed at turning in close to-most advantageous answers within mathematically guaranteed bounds of the genuine foremost [2, 3, 9].

More currently, the emergence of machine learning, especially deep gaining knowledge of, has encouraged researchers to discover records-pushed methods for addressing the complexity of NP-difficult issues [13, 16]. Several strategies were proposed to mix gaining knowledge of-primarily based strategies with classical approximation frameworks by means of both guiding heuristic procedures or studying solution patterns from information [13, 17]. The goal of this paper is to investigate how neural network models can be incorporated into approximation algorithms to clear up complicated combinatorial troubles inclusive of set cowl, graph coloring, and other combinatorial optimization responsibilities more successfully [9, 13, 16].

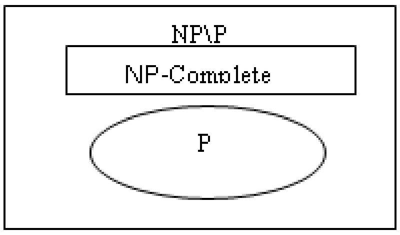

Fig.1 The NP-world as believed by researchers.

The affirmation or refutation of the conjecture P ≠ NP stays one of the most crucial open questions in pc technology and stands as one of the most celebrated unsolved problems in mathematics [7]. The concept of NP-completeness became first mounted by way of Stephen Cook in 1971 [4], with the Boolean satisfiability problem (SAT) being the first trouble established to be NP-entire. SAT asks whether or not there exists a fulfilling mission for a propositional method expressed in conjunctive ordinary form (CNF).

On Once a hassle is proven to be NP-hard, the subsequent step is to decide whether to intention for an precise answer or accept an approximate one. Since it's far unlikely that precise solutions can be located in polynomial time for massive instances, specific methods usually rely upon exhaustive search strategies including department-and-sure algorithms [9], or on formulating the hassle using mathematical programming techniques along with integer linear programming (ILP) or combined-integer programming (MIP) [2, 12]. However, such strategies are computationally high priced and therefore realistic simplest for small-sized trouble times [2, 9].

To find a good-quality solution within a reasonable amount of time, two main types of algorithms are commonly used:

- Approximation algorithms
- Heuristic algorithms

An approximation algorithm is one for which it is possible to mathematically prove how close its solution is to the optimal one—either in the worst-case scenario or on average.

On the other hand, the performance of heuristic algorithms is generally evaluated empirically—by running them multiple times on either randomly generated instances or well-known benchmark problems [9, 12]. Heuristics can be quite simple yet effective, such as dispatching rules in scheduling problems. However, most modern heuristics rely on more sophisticated local search techniques, which include methods like neighborhood search, tabu search, and simulated annealing [9, 13, 14]. These methods are designed to escape local optima and explore the solution space efficiently, often using probabilistic decisions or memory-based strategies to improve solution quality over time.

Approximation algorithms are designed to produce solutions in polynomial time, but this efficiency comes at the cost of optimality. The solutions they generate are guaranteed to be within a fixed performance ratio of the actual optimum [2, 3, 9]. In contrast, heuristic algorithms generate feasible solutions without guarantees on how close those solutions are to the optimal value [9, 12].

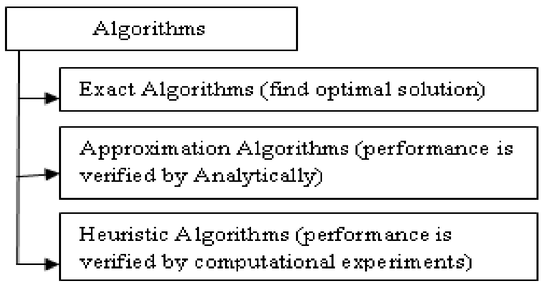
Algorithms
Exact Algorithms (find optimal solution)
Approximation Algorithms (performance is verified by Analytically)
Heuristic Algorithms (performance is verified by computational experiments)

Fig.2 Shows the type of Algorithms

The "goodness" of an approximation algorithm is typically evaluated using the approximation ratio, defined as:

Approximation Ratio=F(SA)F(SOPT)\text{Approximation Ratio} = \frac{F(S_A)}{F(S_{OPT})}Approximation Ratio=F(SOPT)F(SA)

where F(SA)F(S_A)F(SA) is the value of the objective function for the solution SAS_ASA produced by the approximation algorithm, and F(SOPT)F(S_{OPT})F(SOPT) is the value of the objective function for the optimal solution [2, 3].

An algorithm is called an rrr-approximation algorithm if it runs in polynomial time and guarantees that the solution it finds has a value no worse than rrr times the optimum for any instance of the problem. The parameter rrr is known as the worst-case ratio bound [2, 3, 9].

Approximation and Optimization

Complexity theory primarily studies decision problems, which ask whether a solution exists that satisfies specific conditions, with answers restricted to yes or no. The complexity classes NP and NP- complete are defined based on these decision problems.

In contrast, optimization problems focus on finding the best possible solution according to some objective. Every optimization problem has a related decision problem version.

The optimization counterparts of the classes NP, NP-complete, and P are NPO, NP-hard, and PO, respectively.

- NPO is the class of optimization problems whose associated decision problems belong to NP.
- NP-hard consists of optimization problems whose decision versions are NP-complete.
- PO includes optimization problems that can be solved exactly in polynomial time.

Formally, an NPO problem \Phi is defined as a…

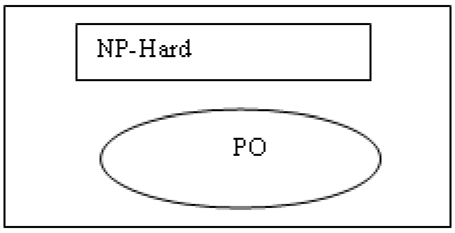

Fig.3 The NP-world as believed by researchers.

- quadruplet (I*Φ*, SOL*Φ*, m*Φ*, opt*Φ*), where:
- I*Φ* draws the set of instances of *Φ*;
- SOL*Φ*(I) draws the set of feasible solutions
- of an instance I □ I*Φ*;
- m*Φ*(I,S) is the measure of a solution S □
- SOL*Φ*(I) (the objective value of S);
- opt*Φ*□{min, max}.

Moreover, any I □ IΦ and S □ SOLΦ(I) are recognizable in time polynomial in |I|, and, for any I
□ IΦ and S □ SOLΦ(I), mΦ(I,S) is computable in time polynomial in |I|. We will denote by optΦ(I) the optimal value (the value of the optimal solution) of I and by ωΦ(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 Φ′= (IΦ′, SOLΦ′, mΦ′, optΦ′) where IΦ′= IΦ, SOLΦ′= SOLΦ and optΦ′ = max if optΦ = min and optΦ′ = min if optΦ = 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 $h(A,I) = |\omega(I) - m(I,S)| / |\omega(I) - opt(I)|$.

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:

These assert that for a specific NP-hard problem, there exists a polynomial-time approximation algorithm whose performance guarantee is bounded above (or bounded below in the case of maximization problems) by a fixed constant factor.

Negative results:

These state that for a given NP-hard problem, no polynomial-time approximation algorithm can achieve a specific approximation ratio unless an unlikely complexity-theoretic assumption (such as P = NP) is true.;

Conditional results:

These results establish relationships either between two types of approximation behaviors for the same problem or between the approximation behaviors of different NP-hard problems. Essentially, these are reductions that preserve certain approximability properties of the problems involved.

Polynomial approximation faces two main challenges:

- Operational: This concerns the need for fast and high-quality solutions to real-world problems, which are essential for the efficient functioning of complex human systems.
- Structural: This relates to the inherent difficulty of the problems themselves.

Because of these challenges, it is often impossible to fully solve certain problems exactly in practice. On the other hand, relying on heuristics that do not always guarantee feasible solutions, or that may not be efficient, can be impractical, and even economically or socially risky.

Polynomial-time approximation algorithms provide a valuable compromise: while they may not always find the best possible solution, they guarantee solutions that satisfy all problem constraints and are as close to optimal as possible. Following classical algorithm theory, the classification of

approximation algorithms can be extended to classify the problems they solve. More specifically, by focusing on the standard approximation ratio (denoted by r), we can classify approximation algorithms based on this ratio, which defines what is called the approximation level.
The most common approximation levels are: Constant ratio: An algorithm is called a constant- ratio algorithm when its approximation ratio r is a fixed constant. This means that the ratio r between the algorithm's solution value and the optimal solution does not depend on any parameter of the input instance I. In other words, the quality guarantee remains the same regardless of the size or specifics of the problem instance.
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 i> 0; for any such ϵ, they compute a feasible solution guaranteeing approximation ratio 1+ϵ (1-ϵ 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/ϵ*;
Logarithmic ratio: when the approximation ratio of the algorithm is O(log|I|);
Polynomial ratio: when $\exists$ *ϵ*>0 such that the approximation ratio achieved is O(n^{1-i}) (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 Ccomplete under reduction R if Φ C and for any other problem Φ′C, Φ′ R-reduces to Φ.

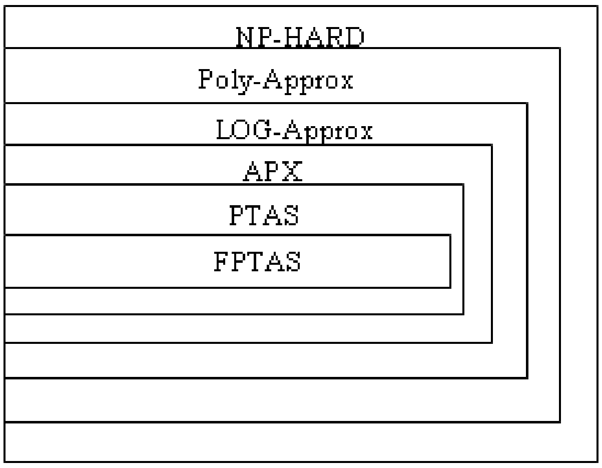

Fig 4. The main approximability classes (under the hypothesis P≠NP) for standard approximation.

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 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]).

DIFFERENT APPROACHES FOR APPROXIMATION

Randomized Approximation Algorithms

A randomized algorithm, also known as a probabilistic algorithm, is one that incorporates randomness into its logic [12, 13]. These algorithms typically use uniformly random bits as auxiliary input to make certain decisions during execution. The intent is to achieve good performance on average, especially in cases where worst-case analysis is too pessimistic [12, 13, 16].

Formally, since the algorithm relies on randomness, its behavior and performance (such as running time or solution quality) are described as random variables. The output or time complexity may vary depending on the sequence of random bits used during a particular run [13, 16].
For a maximization problem, we say that an algorithm AAA is a factor-hhh approximation algorithm if, for every problem instance III with an optimal solution value OPT(I)OPT(I)OPT(I), the expected value of the solution produced by AAA, denoted A(I)A(I)A(I), satisfies:
E[A(I)]≥1h·OPT(I)\mathbb{E}[A(I)] \geq \frac{1}{h} \cdot OPT(I)E[A(I)]≥h1·OPT(I)
This indicates that, on average, the solution is guaranteed to be within a factor hhh of the optimum [2, 3, 9].

Here, E[A(I)]\mathbb{E}[A(I)]E[A(I)] represents the expected performance over the random choices made by the algorithm [2, 3, 9].
The simplicity of design is one of the foremost motivations for using randomized algorithms. In many cases, simple randomized algorithms perform as well as—or even better than—complex deterministic counterparts [12, 13].
Regarding speed, for some problems, the best-known randomized algorithms are significantly faster than the fastest known deterministic algorithms [13, 16].
A key reason for their effectiveness is the presence of many witnesses for "yes" or "no" answers in typical problem instances—making it likely that randomized choices will lead to correct solutions [16].
Randomization also aids in load balancing, especially in distributed and parallel computing systems, where it ensures fair distribution of tasks and avoids bottlenecks [12].
Another powerful technique is random sampling, where a small representative sample is selected from a large input dataset. The algorithm solves the problem on the sample and generalizes the result back to the full dataset. This is particularly useful in large-scale data applications and approximation schemes [13].
In parallel computing, symmetry breaking is critical to avoid deadlock and ensure task differentiation among processors; randomization provides an elegant and efficient way to achieve this [12].
Lastly, the process of derandomization—transforming a randomized algorithm into a deterministic one—is often used to obtain strong performance guarantees. In fact, many of the best deterministic algorithms stem from derandomizing simple and effective randomized algorithms [12, 13].
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.

Online computation

Online algorithms must make decisions step by step without access to the entire input in advance—they receive the input as a sequence and must process it immediately, without knowing future requests. In such uncertain and adversarial environments, incorporating randomness into the algorithm's logic can be highly beneficial. Randomized online algorithms are harder for adversaries to predict or manipulate, as their decisions are not deterministic but depend on random choices [12, 13].
Traditional optimization techniques assume complete knowledge of all input data beforehand. However, in practical situations, this assumption rarely holds. Decisions must often be made incrementally as new data arrives, which motivates the study of online optimization.
An algorithm is online if it must produce a partial solution whenever a new piece of data (request) arrives. Formally, the input is modeled as a finite sequence of requests r_1, r_2, r_3, \dots revealed one at a time. How the input is revealed depends on the chosen model, with the sequence model and time- stamp model being the most common.
Sequence Model: Requests must be handled in the order they appear. When processing request r_j, the online algorithm \text{ALG} has no knowledge of any future requests r_i for i > j. Upon arrival of r_j, ALG must serve it immediately, following problem- specific rules. Serving r_j incurs a cost, and the goal is to minimize the total service cost. Decisions are irrevocable—only after serving r_j is r_{j+1} revealed.
Time-Stamp Model: Each request r_j has a release time t_j > 0, indicating when it becomes available. At any time t, ALG must decide its actions based solely on requests released up to that time. Requests arrive in non-decreasing order of release times, and serving them incurs costs. The objective remains minimizing the total cost.

primal-dual method

The primal-dual method for designing approximation algorithms involves formulating the given problem as a primal integer program and then considering the dual of its linear programming (LP) relaxation. By suitably relaxing certain constraints, this approach can lead to efficient algorithms with provable approximation guarantees for a wide range of NP-hard problems in combinatorial optimization. The key idea is to construct feasible solutions to the primal and dual problems simultaneously, maintaining a relationship between their costs. The method guarantees that the cost of the primal solution obtained is at most α times the cost of the dual solution, implying that the primal result is within a factor of α of the optimal solution [2, 3, 9].

While the dual solution's value is always within a factor of α of the primal, it may be even closer in practice. By evaluating both primal and dual outcomes, one can often obtain a better instance- specific guarantee than α.

The performance guarantee of a primal-dual algorithm is fundamentally tied to the integrality gap of the integer programming formulation. The integrality gap represents the worst-case ratio between the optimal integer solution and the optimal LP relaxation across all instances. Since the performance analysis relies on comparing primal and dual solutions, the guarantee achieved by the algorithm cannot surpass the integrality gap of the chosen formulation.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: $y_j \; > 0 \;\rightarrow\; \sum_i a_{ij}\, xj \;=\; b_j$

Analysis and Results

iThis paper, titled "Approximation Algorithms for NP-Hard Problems," offers a comprehensive discussion of fundamental concepts, methodologies, and practical applications of approximation algorithms. The following tables present a comparative analysis of various approximation techniques applied to a range of NP-hard problems.

Table 1 Comparison of different Approximation Approaches.

Approaches for Approximation Algorithms				
	Performance	**Approximation guarantee**	**Feasibility**	**Resources used**
Randomization	**Performance can be achieved by assigning each vertex with equal probability.** **This gives us a relative performance bound within constant factor 2.** **Use semidefinite programming to do a lot better.** **The reliability of a randomized algorithm can be improved, if more random trials are performed.**	**This is a polynomial- time 2-approximation algorithm means that the solution returned by algorithm is at most twice the size of an optimal.**	**This gives different solutions but all solutions near to optimal.** **Algorithms that are efficient in that their running times are bounded by a polynomial in the size of the input.** **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.**	**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.**

Online Computation	**quality of an on-line algorithm is mostly measured by evaluating its worst case performance.** **The competitive ratio of a specific on-line algorithm is not the answer to this problem.** **A lower bound on the competitive ratio of every possible on- line algorithm answers the question!** **Such lower bounds can be achieved by providing a specific set of instances on which no on-line algorithm can perform well.**	**Numerous problems in operating systems, compilers, graphics, robotics.** **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.**	**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.**	**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	**Solving by adjusting dual variables until primal ones are feasible.** **It reduces the degree of infeasibility of the primal one** **at the same time.**	**The more trials, the better the approximation.**	**No optimal dual solution needed.** **The final dual solution is used as a lower-bound for the optimum solution value by means of weak** **duality.**	**The primal- dual method gives rise to min-max relations which have far-reachibg algorithmatic significance.**
Markov Chain Monte Carlo Method	**1 Even when density is known we may have difficulty in generating** **samples from the distribution corresponding to it.** **For any state of the Markov chain, there is a positive probability of visiting all other states.** **Very complex models can be analyzed.** **Length of the searching phase is difficult to identify.**	**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.** **MCMC is only one way to approximate the posterior distribution. It is not a modeling** **approach itself!**	**MCMC is the general procedure of simulating such Markov chains and using them to draw inference about the characteristics of f(x).** **In the sense that no rigorous guarantees could be given for the quality of the approximate solutions they produced.**	**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.**

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.

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 o n http://www.lamsade.dauphine.fr/cahiers.html.

[2] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti- Spaccamela, and ε. 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] [ε. R. Garey and D. S. Johnson, "Computers and intractability. A guide to the theory of NP-completeness",

[8] H. Freeman, San Francisco, 1979.

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

[10] 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

[11] 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

[12] C. H. Papadimitriou and K. Steiglitz, "Combinatorial optimization: algorithms and complexity", Prentice Hall, New Jersey, 1981.
[13] W.R. Gilks, S. Richardson, and D. J. Spiegelhalter, editors. Markov Chain Monte Carlo in Practice.Chapman and Hall, Suffolk, 1996.
[14] W. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A.
[15] H. Teller, and E. Teller. Equations of state calculations by fast computing machines. Journal of Chemical Physics, 21:1087–1092, 1953.
[16] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. Biometrika, 57:97– 109, 1970.
[17] A. Barker. Monte Carlo calculations of the radial distribution functions for a proton-electron plasma. Australian Journal of Physics, 18:119–133, 1965.