Abstract

In this paper, overview of the techniques used to design and analyze algorithms that provide approximate solutions to NP-hard problems and the various applications of approximation algorithms, which are already in practice, are presented. The methods and algorithms for solving such NP-hard problems, with the help of some approximations and approximation algorithms, are also presented. Some relevant examples are provided to make the point clear.

1.Introduction

In computer science and operations research, approximation algorithms are algorithms used to find approximate solutions to optimization problems. Approximation algorithms are often associated with NP-hard problems; since it is unlikely that there can ever be efficient polynomial-time exact algorithms solving NP-hard problems, one settles for polynomial-time sub-optimal solutions. Unlike heuristics, which usually only find reasonably good solutions reasonably fast, one wants provable solution quality and provable run-time bounds. Ideally, the approximation is optimal up to a small constant factor (for instance within 5% of the optimal solution). Approximation algorithms are increasingly being used for problems where exact polynomial-time algorithms are known but are too expensive due to the input size. A typical example for an approximation algorithm is the one for vertex cover in graphs: find an uncovered edge and add both endpoints to the vertex cover, until none remain. It is clear that the resulting cover is at most twice as large as the optimal one. This is a constant factor approximation algorithm with a factor of 2.

NP-hard problems vary greatly in their approximability; some, such as the bin packing problem, can be approximated within any factor greater than 1 (such a family of approximation algorithms is often called a polynomial time approximation scheme or PTAS). Others are impossible to approximate within any constant, or even polynomial factor unless P = NP, such as the maximum clique problem.

NP-hard problems can often be expressed as integer programs (IP) and solved exactly in exponential time. Many approximation algorithms emerge from the linear programming relaxation of the integer program.

1.1.Approach to attacking NP-hard problems

NP-hard problems cannot be solved in polynomial time. These problems may have practical value but are solvable in exponential time, at best. These problems may be acceptable for small input size. These are possible to isolate special cases that are solvable in polynomial time, possible to find near-optimal solution in polynomial time.We may want to get a solution to those problems in polynomial time. Some solutions may be of sub-exponential complexity, such as 2n=c for c > 1, 2p n, or nlog n. Availability of sub-exponential complexity solution allows for solution of problems with larger data sets $O(n^4)$ complexity may not be good for large data sets; we'll prefer O(n) or $O(n^2)$ complexity algorithms. These give non-optimal solution in polynomial time . These are Often good enough for practical purposes. Such algorithms are Known as approximation algorithms

1.2.Comparison with heuristics

Heuristics find reasonably good solution in a reasonable time. These may be based on isolating important special cases that can be solved in polynomial time. But Heuristics are not guaranteed to work on every instance of the problem data set. Exponential algorithms may still show exponential behavior even when used with heuristics. Approximation algorithms give provable solution quality in provable run-time bounds. Approximation is optimal up to a small constant factor (like 5%). Remove the requirement that the algorithm must always generate an optimal solution and replace it with the requirement that the algorithm must always generate a feasible solution with value close to the value of an optimal (approximate) solution. For NP-hard problems, each optimal solution may not be obtainable in reasonable computation time. A second goal is to get an algorithm that almost always generates optimal solution; such algorithm is called probabilistically good algorithm[1]

1.3. Applications of Approximation Algorithms

1.3.1.Vertex Cover

Problem: Find a vertex-cover of maximum size in a given undirected graph.
This optimal vertex-cover is the optimization version of an NP-complete problem but it is not too hard to find a vertex-cover that is near optimal.

1.3.2.Travelling Salesman Problem

Problem: Given a complete undirected graph G=(V, E) that has nonnegative integer cost c(u, v) associated with each edge (u, v) in E, the problem is to find a hamiltonian cycle (tour) of G with minimum cost.

1.3.3.Absolute approximations

1.3.3.1.Planar graph coloring

Assignment of colors (or labels) to vertices in a graph such that no two adjacent vertices share the same color
Determine the minimum number of colors needed to color a planar graph G = (V;E)

1.3.3.2.Scheduling Independent Tasks:

We consider a set of m identical processors P={P1,P2,...Pm} used for executing the set T={T1,T2,...Tn} of n independent , non-preemptable malleable tasks (MT). Each MT needs for its execution at least 1 processor. The number of processors allotted to a task is unknown in advance.

1.3.4.∑-Approximations:

1.3.4.1.Bin Packing Problem:

In the **bin packing problem**, objects of different volumes must be packed into a finite number of bins or containers each of volume V in a way that minimizes the number of bins used. In computational complexity theory, it is a combinatorial NP-hard problem.

1.3.4.2.Scheduling Independent Tasks

Obtaining minimum finish time schedules on m, m >/ 2 identical processors is NP-hard. There exists a very simple scheduling rule that generates schedules with a finish time very close to that of an optimal schedule. An
instance I of the scheduling problem is defined by a set of n task times, ti, 1</ t </ n, and m, the number of processors.

1.3.5.Polynomial-time approximation scheme (PTAS)

A PTAS is an algorithm which takes an instance of an optimization problem and a parameter $\varepsilon > 0$ and, in polynomial time, produces a solution that is within a factor $1 + \varepsilon$ of being optimal (or $1 - \varepsilon$ for maximization problems). For example, for the Euclidean traveling salesman problem, a PTAS would produce a tour with length at most $(1 + \varepsilon)L$, with L being the length of the shortest tour.

1.3.5.1.Scheduling Independent Tasks

We have seen that the LPT rule leads to a {1/3 - l/{3m))-approximate algorithm for the problem of obtaining an m processor schedule for n tasks. A polynomial time approximation scheme is also known for this problem.

This scheme relies on the following scheduling rule: (i) Let k be some specified and fixed integer. (ii) Obtain an optimal schedule for the k longest tasks. (iii) Schedule the remaining n - k tasks using the LPT rule.

1.3.5.2.PTAS for Knapsack

A smarter approach to the knapsack problem involves brute-forcing part of the solution and then using the greedy algorithm to finish up the rest . In particular, consider all O(knk) possible subsets of objects that have up to k objects, where k is some fixed constant .Then for each subset, use the greedy algorithm to fill up the rest of the knapsack in O(n) time. Pick the most profitable subset A. The total running time of this algorithm is thus

O(knk+1). If O is the optimal subset, then the resulting approximation P(A) achieves P(O) ≤ P(A) (1+ 1/k)

1.3.6.Fully Polynomial Time Approximation Schemes

The approximation algorithms and schemes we have seen so far are particular to the problem considered. There is no set of well defined techniques that one may use to obtain such algorithms. The heuristics used depended

very much on the particular problem being solved. For the case of fully polynomial time approximation schemes, we can identify three underlying techniques. These techniques apply to a variety of optimization problems. We shall discuss these three techniques in terms of maximization problems

1. **Rounding**
2. **Interval Partitioning**
3. **Separation**

1.3.7. Probabilistically Good Algorithms

The approximation algorithms of the preceding sections had the nice property that their worst case performance could be bounded by some constants (kin the case of an absolute approximation and<: in the case of an ∑- approximation). The requirement of bounded performance tends to categorize other algorithms that "usually work well" as being bad. Some algorithms with unbounded performance may in fact "almost always" either solve the problem exactly or generate a solution that is "exceedingly close" in value to the value of an optimal solution. Such algorithms are "good" in a probabilistic sense. If we pick a problem instance I at random then there is a very high probability that the algorithm will generate a very good approximate solution. In this section we shall consider two algorithms with this property. Both algorithms are for NP-hard problems.

2. Literature Survey

2.1. The Primal-Dual Method for Approximation Algorithms

David P. Williamson

IBM T.J. Watson Research Center and

IBM Almaden Research Center

In this survey, they give an overview of a technique used to design and analyze algorithms that provide approximate solutions to NP-hard problems in combinatorial optimization. Because of parallels with the primal-dual method commonly used in combinatorial optimization, we call it the

primal-dual method for approximation algorithms. We show how this technique can be used to derive approximation algorithms for a number of different problems, including network design problems, feedback vertex set problems, and facility location problems.

2.2. A ½ factor approximation algorithm for two-stage stochastic matching problems

Nan Kong, Andrew J. Schaefer *

Department of Industrial Engineering, University of Pittsburgh, 1048 Benedum Hall, Pittsburgh, PA15261, USA

Received 26 November 2003; accepted 22 October 2004

They introduce the two-stage stochastic maximum-weight matching problem and demonstrate that this problem is NP-complete. They give a factor 1. 2 approximation algorithm and prove its correctness. They also provide a tight example to show the bound given by the algorithm is exactly 12. Computational results on some two-stage stochastic bipartite matching instances indicate that the performance of the approximation algorithm appears to be substantially better than its worst-case performance.

2.3. Simpler and Better Approximation Algorithms for NetworkDesign

Anupam Gupta Amit Kumar, Tim Roughgarden

They give simple and easy-to-analyze randomized approximation algorithms for several well-studied NP-hard network design problems. Their algorithms improve over the previously best known approximation ratios.

2.4. Approximation Algorithms for Array Partitioning Problems

S. Muthukrishnan, Torsten Suel

They study the problem of optimally partitioning a two-dimensional array of elements by cutting each coordinate axis into p*q rectangular regions. This problem arises in several applications in databases, parallel computation, and image processing. Their main contribution are new approximation algorithms for these NP-Complete problems that improve significantly over previously known bounds. The algorithms are fast and simple, work for a variety of measures of partitioning quality, generalize to dimensions d>2 , and achieve almost optimal approximation ratios. They also extend previous NP-Completeness results for this class of problems.

2.5. Approximation Algorithms for Orienteering and Discounted-Reward TSP

Avrim Blum, Shuchi Chawla, David R. Karger§ Terran Lane, Adam Meyersonk

Maria Minkoff

In this paper, they give the first constant-factor approximation algorithm for the rooted ORIENTEERING problem, as well as a new problem that they call the DISCOUNTED-REWARD-TSP, motivated by robot navigation. In both problems, they are given a graph with lengths on edges and rewards on nodes, and a start nodes. In the ORIENTEERING problem, the goal is to find a path starting at s that maximizes the reward collected, subject to a hard limit on the total length of the path. In the DISCOUNTEDREWARD- TSP, instead of a length limit they are given a discount factor γ, and the goal is to maximize

2.6. A General Greedy Approximation Algorithm with Applications

Tong Zhang
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598
tzhang@watson.ibm.com

Greedy approximation algorithms have been frequently used to obtain sparse solutions to learning problems. In this paper, they present a general greedy algorithm for solving a class of convex optimization problems. They derive a bound on the rate of approximation for this algorithm, and show that our algorithm includes a number of earlier studies as special cases.

2.7.Approximation Algorithms for Metric Facility Location Problems

Mohammad Mahdiany Yinyu Yez Jiatheyi Zhang x

In this paper they present a 1.52-approximation algorithm for the metric uncapacitated facility location problem, and a 2-approximation algorithm for the metric capacitated facility location problem with soft capacities. Both these algorithms improve the best previously known approximation factor for the corresponding problem, and our soft-capacitated facility location algorithm achieves the integrality gap of the standard LP relaxation of the problem. Furthermore, they will show, using a result of Thorup, that our algorithms can be implemented in quasi-linear time.

2.8.Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming

MIC13EL X. GOEMANS
Massachusetts Institute of Technology, Cambridge, Massachusetts
AND
DAVID P. WILLIAMSON
IBM T. J. Watson Research Center, Yorktown Heights, New York

They present randomized approximation algorithms for the maximum cut (MAX CUT) and maximum 2-satisfiability (MAX 2SAT) problems that always deliver solutions of expected value at least .87856 times the optimal value. These algorithms use a simple and elegant technique that randomly rounds the solution to a nonlinear programming relaxation.

2.9. A Constant Factor Approximation Algorithm for Generalized Min-Sum Set Cover

Anupam Guptay Ravishankar Krishnaswamyyz

Consider the following generalized min-sum set cover or multiple intents re-ranking problem proposed by Azar et al. (STOC 2009). They are given a universe of elements and a collection of subsets, with each set S having a

covering requirement of K(S). The objective is to pick one element at a time such that the average covering time of the sets is minimized, where the covering time of a set S is the ¯rst time at which K(S) elements from it have been selected. There are two well-studied extreme cases of this problem: (i) when K(S) = 1 for all sets, they get the min-sum set cover problem, and (ii) when K(S) = (S) for all sets, they get the minimum-latency set cover problem. Constant factor approximations are known for both these problems. In their paper, Azar et al. considered the general problem and gave a logarithmic approximation algorithm for it. In this paper, they improve their result and give a simple randomized constant factor approximation algorithm for the generalized min-sum set cover problem.

2.10. A Local 2-approximation Algorithm for the Vertex Cover Problem

Matti _Astrand, Patrik Flor_een, Valentin Polishchuk, Joel Rybicki,
Jukka Suomela, and Jara Uitto
Helsinki Institute for Information Technology HIIT, University of Helsinki
P.O. Box 68, FI-00014 University of Helsinki, Finland
ffirstname.lastnameg@cs.helsinki.fi

They present a distributed 2-approximation algorithm for the minimum vertex cover problem. The algorithm is deterministic, and it runs in (t + 1)2 synchronous communication rounds, where t is the maximum degree of the graph. For _ = 3, they give a 2-approximation algorithm also for the weighted version of the problem.

2.11. Algorithms and applications for approximate nonnegative matrix factorization

MichaelW. Berrya, Murray Brownea, Amy N. Langvilleb,1, V. Paul Paucac,2,
Robert J. Plemmonsc,2
aDepartment of Computer Science, University of Tennessee, Knoxville, TN 37996-3450, USA
bDepartment of Mathematics, College of Charleston, Charleston, SC 29424-0001, USA
cDepartments of Computer Science and Mathematics, Wake Forest University, Winston-Salem, NC 27109, USA
Available online 29 November 2006

The development and use of low-rank approximate nonnegative matrix factorization (NMF) algorithms for feature extraction and identification in the fields of text mining and spectral data analysis are presented. The evolution and convergence properties of hybrid methods based on both sparsity and smoothness constraints for the resulting nonnegative matrix factors are discussed. The interpretability of NMF outputs in specific contexts are provided along with opportunities for future work in the modification of NMF algorithms for large-scale and time-varying data sets.

3.Methodologies used

3.1.Vertex Cover

Definition: A vertex-cover of an undirected graph G=(V, E) is a subset of V`subset of V such that if edge (u, v) is an edge of G then either u in V or v in V` (or both).

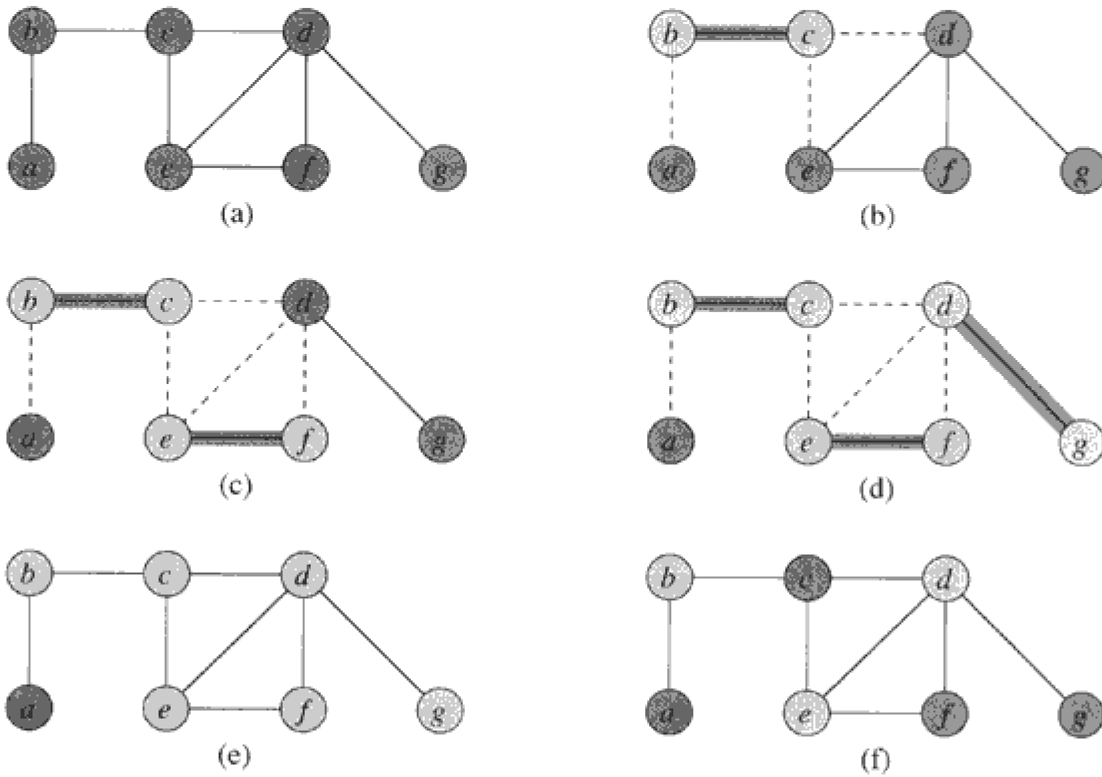
Problem: Find a vertex-cover of maximum size in a given undirected graph.

This optimal vertex-cover is the optimization version of an NP-complete problem but it is not too hard to find a vertex-cover that is near optimal.

APPROX-VERTEX_COVER (G: Graph)

1. c ← { }
2. E` ← E[G]
3. while E` is not empty do
4. Let (u, v) be an arbitrary edge of E`
5. c ← c U {u, v}
6. Remove from E` every edge incident on either u or v
7. return c

Example

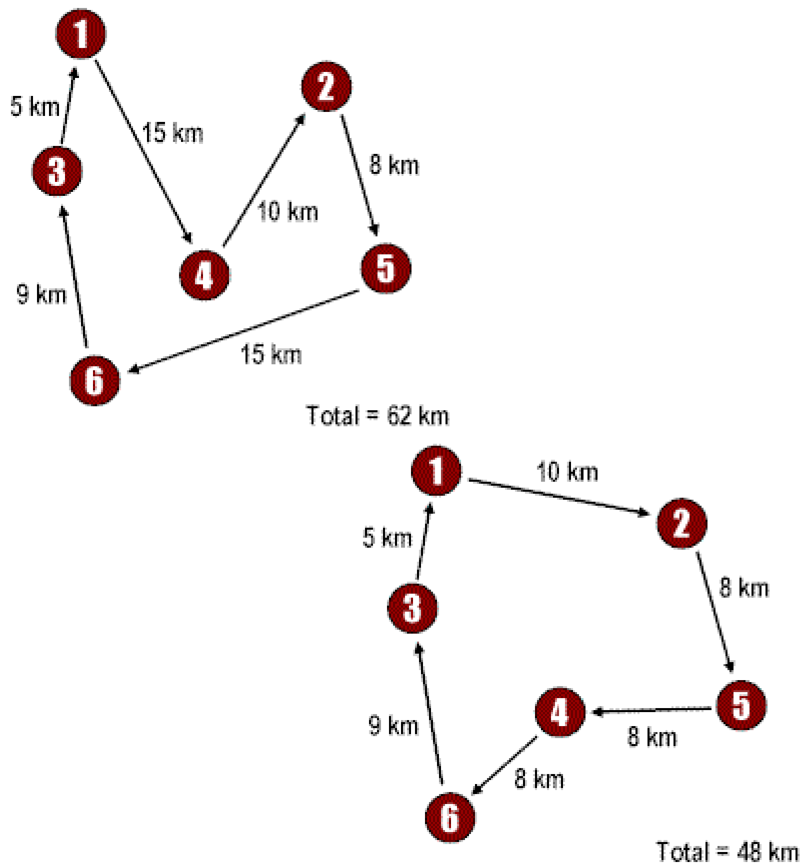


Analysis

It is easy to see that the running time of this algorithm is O(V+E), using adjacency list to represent E`.

3.2.Travelling Salesman Problem

Problem: Given a complete undirected graph G=(V, E) that has nonnegative integer cost c(u, v) associated with each edge (u, v) in E, the problem is to find a hamiltonian cycle (tour) of G with minimum cost.



A salespersons starts from the city 1 and has to visit six cities (1 through 6) and must come back to the starting city i.e., 1. The first route (left side) 1→ 4 → 2 → 5 → 6 → 3 → 1 with the total length of 62 km, is a relevant selection but is not the best solution. The second route (right side) 1 → 2 → 5 → 4 → 6 → 3 → 1 represents the must better solution as the total distance, 48 km, is less than for the first route.

Suppose c(A) denoted the total cost of the edges in the subset A subset of E i.e.,

$$c(A) = \sum_{u,v \text{ in } A} c(u, v)$$

Moreover, the cost function satisfies the triangle inequality. That is, for all vertices u, v, w in V, w have $c(u, w) \leq c(u, v) + c(v, w)$.

Note that the TSP problem is NP-complete even if they require that the cost function satisfies the triangle inequality. This means that it is unlikely that they can find a polynomial-time algorithm for TSP.

3.2.1.TSP with the Triangle-Inequality

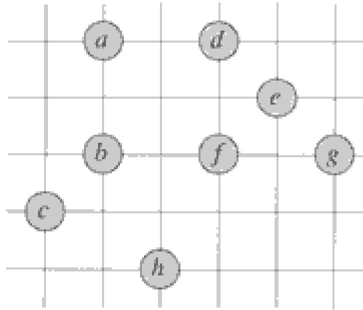
When the cost function satisfies the triangle inequality, they can design an approximate algorithm for TSP that returns a tour whose cost is not more than twice the cost of an optimal tour.

3.2.2.Outline of an APPROX-TSP-TOUR

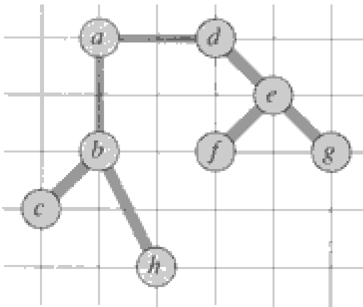
First, compute a MST (minimum spanning tree) whose theyight is a lotheyr bound on the length of an optimal TSP tour. Then, use MST to build a tour whose cost is no more than twice that of MST's theyight as long as the cost function satisfies triangle inequality.

3.2.3.Operation of APPROX-TSP-TOUR

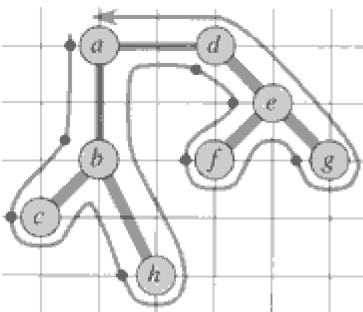
Let root r be a in following given set of points (graph).



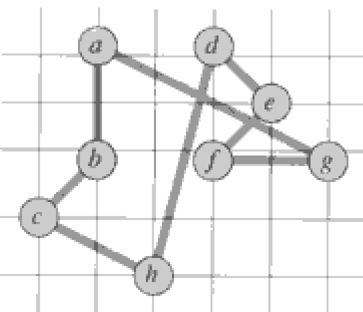
Construct MST from root a using MST-PRIM (G, c, r).



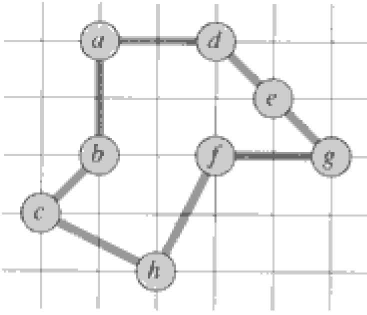
List vertices visited in preorder walk. L = {a, b, c, h, d, e, f, g}



Return Hamiltonian cycle.



Optimal TSP tour for a given problem (graph) would be



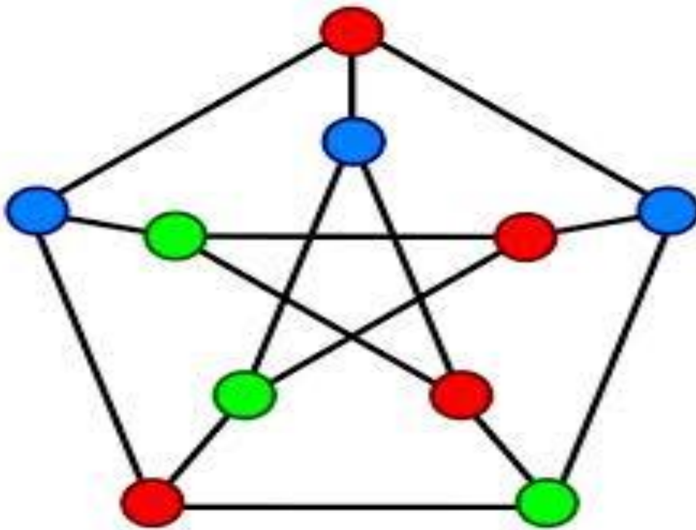
which is about 23% shorter.[2]

3.3.Absolute approximations

3.3.1.Planar graph coloring

Assignment of colors (or labels) to vertices in a graph such that no two adjacent vertices share the same color

1. Determine the minimum number of colors needed to color a planar graph G = (V;E)
2. A graph is 0-colorable iff V =Ø ;
3. A graph is 1-colorable iff E = Ø ;
4. A graph is 2-colorable iff it is bipartite
5. Determining whether a graph is 3-colorable is NP-hard
6. It is known that every planar graph is four colorable
7. It is easy to obtain an algorithm with /Å (I) - ^ Â(I)/ 1



```
algorithm acolor ( V, E )
{
// Determine an approximation to the minimum number of colors
// Input: Set of vertices and set of edges
// Output: Number of colors
if ( V == NULL ) return 0;
if ( E == NULL ) return 1;
if ( G is bipartite ) return 2;
return ( 4 );
}
```

They can determine that a graph is bipartite in time O(/V/ + /E/)

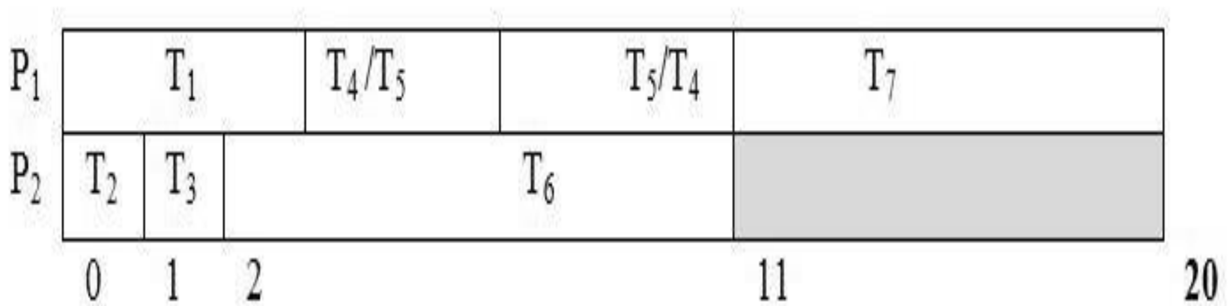
Complexity of acolor is O(/V/ + /E/)

3.3.2.Scheduling Independent Tasks:

They consider a set of m identical processors P={P1,P2,...Pm} used for executing the set T={T1,T2,...Tn} of n independent , non-preemptable malleable tasks (MT). Each MT needs for its execution at least 1 processor. The number of processors allotted to a task is unknown in advance.

Now, the problem may be stated as the one of finding a schedule of minimum length w, provided that the processing spedd functions of the tasks are all concave. Let us note, that this is a realistic case, more often appearing in practice. They will denote this minimum value as w*m.

As far as processors are concerned, function fi are discrete. Hotheyver, in general, it would be also possible that these functions are continuous. In what follows they will use the results of optimal continuous resource allocation to construct good processor schedules.



T1	T2	T3	T4	T5	T6	T7
3	1	1	2	2	6	9

[3]

3.4.∑-Approximations:

3.4.1.Bin Packing Problem:

In the **bin packing problem**, objects of different volumes must be packed into a finite number of bins or containers each of volume V in a way that minimizes the number of bins used. In computational complexity theory, it is a combinatorial NP-hard problem.

There are many variations of this problem, such as 2D packing, linear packing, packing by theyight, packing by cost, and so on. They have many applications, such as filling up containers, loading trucks with theyight capacity constraints, creating file backups in media and technology mapping in Field-programmable gate array semiconductor chip design.

The bin packing problem can also be seen as a special case of the cutting stock problem. When the number of bins is restricted to 1 and each item is characterised by both a volume and a value, the problem of maximising the value of items that can fit in the bin is known as the knapsack problem.

Despite the fact that the bin packing problem has an NP-hard computational complexity, optimal solutions to very large instances of the problem can be produced with sophisticated algorithms. In addition, many heuristics have been developed: for example, the **first fit algorithm** provides a fast

but often non-optimal solution, involving placing each item into the first bin in which it will fit. It requires **Θ**(n log n) time, where n is the number of elements to be packed. The algorithm can be made much more effective by first sorting the list of elements into decreasing order (sometimes known as the first-fit decreasing algorithm), although this still does not guarantee an optimal solution, and for longer lists may increase the running time of the algorithm. It is known, hotheyver, that there always exists at least one ordering of items that allows first-fit to produce an optimal solution.

A variant of bin packing that occurs in practice is when items can share space when packed into a bin. Specifically, a set of items could occupy less space when packed together than the sum of their individual sizes. This variant is known as VM packing[2] since when virtual machines (VMs) are packed in a server, their total memory requirement could decrease due to pages shared by the VMs that need only be stored once. If items can share space in arbitrary ways, the bin packing problem is hard to even approximate. Hotheyver, if the space sharing fits into a hierarchy, as is the case with memory sharing in virtual machines, the bin packing problem can be efficiently approximated.

Formal statement

Given a bin S of size V and a list of n items with sizes $a_1, \dots, a_n$ to pack, find an integer number of bins B and a B-partition $S_1 \cup \dots \cup S_B$ of the set $\{1, \dots, n\}$ such that $\sum_{i \in S_k} a_i \leq V$ for all $k = 1, \dots, B$. A solution is optimal if it has minimal B. The B-value for an optimal solution is denoted **OPT** below. A possible Integer Linear Programming formulation of the problem is:

minimize $B = \sum_{i=1}^{n} y_i$

subject to $\sum_{j=1}^{n} a_j x_{ij} \leq V y_i, \forall i \in \{1, \dots, n\}$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad \forall j \in \{1, \dots, n\}$$

$$y_i \in \{0,1\}, \qquad \forall i \in \{1, \dots, n\}$$

$$x_{ij} \in \{0,1\}, \qquad \forall i \in \{1, \dots, n\} \, \forall j \in \{1, \dots, n\}$$

where $y_i = 1$ if bin i is used and $x_{ij} = 1$ if item j is put into bin i.[3]

First-fit algorithm[edit]

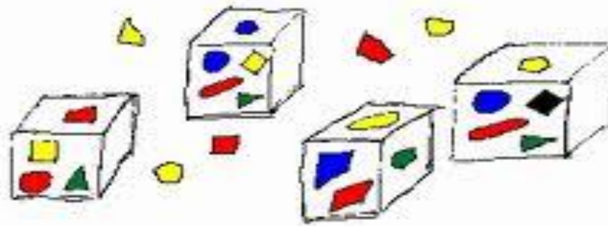
This is a very straightforward greedy approximation algorithm. The algorithm processes the items in arbitrary order. For each item, it attempts to place the item in the first bin that can accommodate the item. If no bin is found, it opens a new bin and puts the item within the new bin.

It is rather simple to show this algorithm achieves an approximation factor of 2, that is, the number of bins used by this algorithm is no more than twice the optimal number of bins. This is due to the observation that at any given time, it is impossible for 2 bins to be at most half full. The reason is that if it would be possible, it would mean that at some point exactly one bin was at most half full and a new one has been opened to accommodate an item of size at most V/2. But since the first one has at least a space of V / 2, the algorithm will not open a new bin for any item whose size is at most V / 2.

Only after the bin fills with more than V / 2 or if an item with a size larger than V / 2 arrives, the algorithm may open a new bin.

Thus if they have B bins, at least B − 1 bins are more than half full. Therefore $\sum_{i=1}^{n} a_i > \frac{B-1}{2}V$.

Because $\frac{\sum_{i=1}^{n} a_i}{V}$ is a lotheyr bound of the optimum value OPT, they get thatB − 1 < 2OPT and therefore B ≤ 2OPT.[4] See the analysis below for better approximation results.



3.4.2.Scheduling Independent Tasks

Obtaining minimum finish time schedules on m, m >/ 2 identical processors is NP-hard. There exists a very simple scheduling rule that generates schedules with a finish time very close to that of an optimal schedule. An

instance I of the scheduling problem is defined by a set of n task times, ti, 1</ t </ n, and m, the number of processors. The scheduling rule they are about to describe is known as the LPT (longest processing time) rule.

Definition An LPT schedule is one that is the result of an algorithm which, 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. An LPT schedule is a schedule that results from this rule[5]

3.5.Polynomial-time approximation scheme

In computer science, a **polynomial-time approximation scheme** (**PTAS**) is a type of approximation algorithm for optimization problems (most often, NP-hard optimization problems).A PTAS is an algorithm which takes an instance of an optimization problem and a parameter ε > 0 and, in polynomial time, produces a solution that is within a factor 1 + ε of being optimal (or 1 - ε for maximization problems). For example, for the Euclidean traveling salesman problem, a PTAS would produce a tour with length at most (1 + ε)L, with L being the length of the shortest tour. The running time of a PTAS is required to be polynomial in n for every fixed ε but can be different for different ε. Thus an algorithm running in time $O(n^{1/\varepsilon})$ or even $O(n^{\exp(1/\varepsilon)})$ counts as a PTAS.

3.5.1.Scheduling Independent Tasks

They have seen that the LPT rule leads to a {1/3 - l/{3m))-approximate algorithm for the problem of obtaining an m processor schedule for n tasks. A polynomial time approximation scheme is also known for this problem.

This scheme relies on the following scheduling rule: (i) Let k be some specified and fixed integer. (ii) Obtain an optimal schedule for the k longest tasks. (iii) Schedule the remaining n - k tasks using the LPT rule.

3.5.2.PTAS for Knapsack

A smarter approach to the knapsack problem involves brute-forcing part of the solution and then using the greedy algorithm to finish up the rest . In particular, consider all O(knk) possible subsets of objects that have up to k objects, where k is some fixed constant [1].Then for each subset, use the greedy algorithm to fill up the rest of the knapsack in O(n) time. Pick the most profitable subset A. The total running time of this algorithm is thus

O(knk+1). If O is the optimal subset, then the resulting approximation P(A) achieves P(O) ≤ P(A) (1+ 1/k)

3.6.FULLY POLYNOMIAL TIME APPROXIMATION SCHEMES

The approximation algorithms and schemes they have seen so far are par·ticular to :he problem considered. There is no set of theyll defined techniques that one may use to obtain such algorithms. The heuristics used depended

very much on the particular problem being solved. For the case of fully polynomial time approximation schemes, they can identify three underlying techniques. These techniques apply to a variety of optimization problems. They shall discuss these three techniques in terms of maximization problems

3.6.1.Rounding

The aim of rounding is to start from a problem instance, I, formulated as in (12.1) and to transform it to another problem instance I' that is easier to solve. This transformation is carried out in such a way that the optimal solution value of I' is "close" to the optimal solution value of I. In particular, if they are provided with a bound, 1:, on the fractional difference bettheyen the exact and approximate solution values then they require that I F*(J) - F*(l')I F*(l) I ~ 1:, where F*(l) and F*(l') represent the optimal solution values of I and I' respectively. I' is obtained from /by changing the objective function to max E q;x;. Since I and I' have the same constraints, they have the same feasible solutions. Hence, if the p;'s and q;'s differ by only a "small" amount, value of an optimal solution to/' will be close to the value of an optimal solution to I.

3.6.2.Interval Partitioning

Unlike rounding, interval partitioning does not transform the original problem instance into one that is easier to solve. Instead, an attempt is made to solve the problem instance I by generating a restricted class of the feasible

assignmen t s £i or sco) , s 0 >, ... , s(n) . Let p ; b e the maxi. mum "-"'J'1= 1 p1x1 amongst all feasible assignments generated for sen . Then the profit interval (0, P1] is divided into subintervals each of size P;El(n - 1) (except possibly the last interval which may be a little smaller). All feasible assignments in S(i) with Ej= 1 p1x1 in the same subinterval are regarded as having the same Ej.1 p1x1 and the dominance rules are used to discard all but one of them. The S Cil resulting from this elimination is used in the generation of sc1+ 1>. Since the number of subintervals for each S(i) is at most r n/€ l + 1, IS(ill :s r n/€ l + 1. Hence, ETIS(ill = O(n 21€). The error introduced in each feasible assignment due to this elimination in S c1> is less than the subinterval length. This error may hotheyver propagate from so> up through sen>. Hotheyver, the error is additive. Let F(I) be the value of the optimal generated using interval partitioning, and F*(J) the value of a true optimal. It follows that F*(J) - F(I) :s (€ ~g P.)l(n - 1). Since P1 :s F*(I), it follows that (F*(I) - F(I))I F*(I) :s E, as desired. In many cases the algorithm may be speeded by starting with a good estimate, LB for F*(I) such that F*(I) ~ LB. The subinterval size is then LB·El(n - 1) rather than P1d(n - 1). When a feasible assignment with value greater than LB is discovered, the subinterval size can be chosen as described above.

3.6.3.Separation

Assume that in solving a problem instance I, they have obtained an sen with feasible solutions having the following Eis.;s,. p1x1 : 0, 3.9, 4.1, 7.8, 8.2, 11.9, 12.1. Further assume that the interval size P,.d(n - 1) is 2. Then the subintervals are [O, 2), [2, 4), [4, 6), [6, 8), [8, 10), [10, 12) and [12, 14) . Each value above falls in a different subinterval and so no feasible assignments are eliminated. Hotheyver, there are three pairs of assignments with values within P;El(n - 1). If the dominance rules are used for each pair, only 4 assignments will remain. The error introduced is at most P;El(n - 1). More formally, let ao, a1 , a2 , ••• , a, be the distinct values of E5_ 1 p1x1 in S(i). Let us assume a0 < a1 < a2 • • • < a,. They will construct a new set J from S(i) by making a left to right scan and retaining a tuple only if its value exceeds the value of the last tuple in J by more than P,.d(n - 1).[7]

3.7.PROBABILISTICALLY GOOD ALGORITHMS

The approximation algorithms of the preceding sections had the nice property that their worst case performance could be bounded by some constants (kin the case of an absolute approximation and<: in the case of an

<:-approximation). The requirement of bounded performance tends to categorize other algorithms that "usually work theyll" as being bad. Some algorithms with unbounded performance may in fact "almost always"

either solve the problem exactly or generate a solution that is "exceedingly close" in value to the value of an optimal solution. Such algorithms are "good" in a probabilistic sense. If they pick a problem instance I at random

then there is a very high probability that the algorithm will generate a very good approximate solution. In this section they shall consider two algorithms with this property. Both algorithms are for NP-hard problems.

4. Gaps in Study

In this paper, I have attempted to cover most of the applications of the approximation problems, their solutions but still some of the problems are left such as:

Non-negative Matrix Factorization,

5. Summary

Approximation algorithms are often associated with NP-hard problems; since it is unlikely that there can ever be efficient polynomial-time exact algorithms solving NP-hard problems, one settles for polynomial-time sub-optimal solutions. Unlike heuristics, which usually only find reasonably good solutions reasonably fast, one wants provable solution quality and provable run-time bounds. Ideally, the approximation is optimal up to a small constant factor (for instance within 5% of the optimal solution). Approximation algorithms are increasingly being used for problems where exact polynomial-time algorithms are known but are too expensive due to the input size.

6. Future Scope

In this paper, an attempt has been made to cover most of the major concepts of approximation algorithms but still some problems couldn’t be covered. These problems will be trued to be covered in future papers.

References:

1.The Primal-Dual Method for Approximation Algorithms
David P. Williamson

2. A ½ factor approximation algorithm for two-stage stochastic matching problems
Nan Kong, Andrew J. Schaefer

3. Simpler and Better Approximation Algorithms for Network Design
Anupam Gupta, Amit Kumar, Tim Roughgarden

4. Approximation Algorithms for Array Partitioning Problems
S. Muthukrishnan, Torsten Suel

5. Approximation Algorithms for Orienteering and Discounted-Reward TSP
Avrim Blum† Shuchi Chawla , David R. Karger, Terran Lane, Adam Meyersonk
Maria Minkoff

6. A General Greedy Approximation Algorithm with Applications
Tong Zhang

7. Approximation Algorithms for Metric Facility Location Problems
Mohammad Mahdiany Yinyu Yez Jiatheyi Zhang x

8. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming

MIC13EL X. GOEMANS, DAVID P. WILLIAMSON

9.A Constant Factor Approximation Algorithm for Generalized Min-Sum Set Cover
Anupam Guptay Ravishankar Krishnaswamyyz
Applications of Approximation Algorithms to
Cooperative Games
Kamal Jain, Vijay V. Vazirani

10. Algorithms and applications for approximate nonnegative matrix factorization
MichaelW. Berrya,, Murray Brownea, Amy N. Langvilleb,1, V. Paul Paucac,2,
Robert J. Plemmonsc,
aDepartment of Computer Science, University of Tennessee, Knoxville, TN 37996-3450, USA
bDepartment of Mathematics, College of Charleston, Charleston, SC 29424-0001, USA
cDepartments of Computer Science and Mathematics, Wake Forest University, Winston-Salem, NC 27109, USA
Available online 29 November 2006

11. Fundamentals of Computer Algorithms
Ellis Horowitz, Sartaj Sahni, Sanguthevar Rajasekaran

12. The Vertex Cover Algorithm
Ashay Dharwadker