Approximation Schemes for Minimum Latency Problems

Sanjeev Arora *
Princeton University

George Karakostas †
Princeton University

Abstract

The *minimum latency problem*, also known as *traveling repairman problem* [1], is a variant of the TSP in which the starting node of the tour is given and the goal is to minimize the sum of the *arrival times* at the other nodes. We present a quasipolynomial-time approximation scheme for this problem when the instance is a weighted tree and when the the nodes lie in $\Re^d$ for some fixed d. The best polynomial time approximation algorithm due to Goemans and Kleinberg, computes a 3.59... approximation.

1 Introduction

The *minimum latency problem*, also known as *traveling repairman problem* [1], is a variant of the TSP in which the starting node of the tour is given and the goal is to minimize the sum of the *arrival times* at the other nodes. (The arrival time is the distance covered before reaching that node.) This problem has a reputation for being much harder than the TSP. An efficient algorithm is known when the nodes are connected by a path graph [1], but the problem is conjectured to be NP-hard for trees, and even caterpillars, i.e., paths with edges sticking out. (The conjectures are attributed to D. West in [9].) By contrast, the TSP can be optimally solved on a tree. However, both problems are known to be NP-hard when the points lie in a general metric space or even the (Euclidean) plane.

Attention has therefore focussed on approximation algorithms. The metric case of the latency problem is MAX-SNP-hard (this follows from the reduction that proves the MAX-SNP-hardness of TSP$(1,2)$ [11]), and therefore the results of Arora et al. [3] imply that unless P = NP, a polynomial time approximation scheme (PTAS) is unlikely to exist. Blum, Chalasani, Coppersmith, Pulleyblank, Raghavan and Sudan gave a 144-approximation algorithm for the metric case and a 8-approximation for weighted trees. Goemans and Kleinberg [7] then gave a 21.55-approximation in the metric case. The Goemans-Kleinberg algorithm requires as a subroutine a good approximation algorithm for the k-TSP problem ("given n nodes and a number k, find the shortest salesman tour containing k nodes"). Improved approximation algorithms for the k-TSP in metric spaces due to Garg [6] and for constant-dimensional Euclidean spaces due to Arora [2] (for planar instances, see also Mitchell [10]) can be used to improve the approximation ratio achieved by the Goemans-Kleinberg algorithm in these cases: 10.78-approximation in the metric case and 3.59-approximation in the Euclidean case.

This paper presents a new and very simple technique that leads to *approximation schemes* for minimum latency for all weighted trees and constant-dimensional Euclidean spaces. To compute a $(1+\epsilon)$-approximation for the problem on n nodes, the algorithm requires $n^{O(\log n/\epsilon)}$ time on weighted trees and $n^{O(\log n \log\log n/\epsilon^2)}$ time in $\Re^2$. We also present a 17.24-approximation in the metric case. Though this approximation ratio is worse than that of the algorithm in [7], our algorithm seems simpler.

As observed by many researchers, one of the complications in the minimum latency problem is the nonlocal nature of the objective function. (This seems to be a problem even in the tree case; see Section 3.) An extra edge inserted in the beginning of the tour affects the latency of *all* the remaining nodes. Previous papers have taken the approach of computing solutions to a sequence of k-TSP instances, for $k = 1, 2, \ldots, n$, and concatenating some of these tours to get the final tour. The main intuition in these algorithms may be described as: visit the nodes closest to the start node as soon as possible.

The main idea in our algorithm is easier to state. The algorithm finds the tour as a union of $O(\log n/\epsilon)$ tours, containing $n_1, n_2, \ldots$ nodes. The important difference is that the choice of $n_1, n_2, \ldots,$ does not depend on the instance; it depends only on n, ϵ. (Thus, as noted in Section 2.1, our algorithm can be viewed as an

*Research supported by NSF CAREER award NSF CCR-9502747, an Alfred Sloan Fellowship, and a Packard Fellowship.
email:arora@cs.princeton.edu

†Research supported under NSF CAREER award NSF CCR-9502747, an Alfred Sloan Fellowship, and a Packard Fellowship.
email:gk@cs.princeton.edu

approximation-preserving reduction to a version of vehicle routing.) In fact, in the metric case, the first salesman tour contains more than $n/2$ nodes. This seems to go against the received intuition of "visit the nodes closest to the start node first." Conceivably, our technique could be combined with that intuition to get better algorithms, but we do not currently know how.

Search ratio of graphs. Koutsoupias, Papadimitriou, and Yannakakis[9] consider a graph exploration problem, in which an explorer is presented with a weighted graph on n nodes. One of the nodes contains a treasure, which the explorer will recognize only when he sees it. The goal is to design a walk on the graph such that the explorer arrives at the treasure as quickly as possible. The *search ratio* of the graph is the worst-case ratio of the arrival time and the distance of the treasure to the start node. The *randomized search ratio* is defined similarly, except the walk may be randomized and so we need the *expected* arrival time at the treasure. As shown in [9], the minimum latency problem is the polyhedral separation problem of the dual of the randomized search ratio problem.

Using the general framework for convex optimization in Grötschel, Lovász, and Schrijver [8], É. Tardos has observed [9] that if the minimum latency problem has a polynomial time approximation scheme for a certain class of metrics, then the randomized search ratio problem has an approximation scheme for that same class of metrics. Thus our algorithm implies the existence of a quasipolynomial time approximation scheme for the randomized search ratio for trees and Euclidean spaces. We do not know of a previous use of the framework in [8] to design an approximation scheme.

1.1 Definitions

We list the definitions of the various problems mentioned above.

Minimum latency: Given : Set of n nodes with $d(i,j)$ denoting the distance between nodes i, j.
Goal: Find a permutation π of nodes such that $\pi(1) = 1$ and we minimize

$$\sum_{i=2}^{n} \sum_{j=1}^{i-1} d(\pi(j), \pi(j+1)).$$

Search ratio: Given : Graph G with distances on edges, and a root vertex r.
Goal: Find

$$\sigma(G,r) = \min_{\pi} \max_{v \in G} \frac{d_\pi(r,v)}{d(r,v)},$$

where $d(r,v)$ denotes the distance from r to v and $d_\pi(r,v)$ denotes the distance from r to v in the walk π. The *randomized search ratio* is defined similarly, except we are allowed a randomized walk, i.e., a distribution over walks, and we replace $d_\pi(r,v)$ by $E[d_\pi(r,v)]$, the expected distance from r to v in the random walk.

Graph metric: Any weighted undirected graph can be used to define a metric space by letting $d(i,j)$ denote the distance from i to j in the graph. We will be especially interested in *tree metrics*, which are derived from weighted trees.

2 The main idea: Local structure does not matter

It is well-known that minimizing the total tour length may give a tour of very high latency. In this section we show that the strategy of minimizing tour lengths works so long as it is done in a local fashion. Namely, to find a $(1+\epsilon)$-approximate minimum latency tour, it suffices to find the tour as a union of $O(\log n/\epsilon)$ segments, where the number of nodes in successive segments decreases geometrically. Within each segment the order of visits to the nodes does not matter, as long the total length is close to minimum. Of course, we have not specified thus far how to partition nodes into the segments in the correct way. In the Euclidean and tree-metric cases, we can do this with a simple dynamic programming. In the metric case, we recourse to a greedy strategy that introduces another source of suboptimality.

We note that the idea of finding a low latency tour as a union of salesman tours/paths is present in all earlier papers [4, 7, 9]. However, those earlier strategies decided in an adaptive fashion which saleman tours to combine. In contrast, our algorithm can decide at the very start how many nodes must be present in each salesman path.

Now we describe our idea. First we point out that it suffices to assume wlog that the minimum nonzero internode distance is 1 and maximum internode distance is $O(n^2/\epsilon)$. The reason is that if L denotes the diameter of the space, then L is a lowerbound on the minimum latency. Let us merge all pairs of nodes with internode distance at most $\epsilon L/n^2$. This affects the latency of the optimum tour by at most ϵL. Furthermore, the ratio of the maximum internode distance and the minimum nonzero internode distance is $O(n^2/\epsilon)$. Since ϵ is constant, we will often think of the maximum internode distance as $O(n^2)$.

Now we describe the break-up into segments. Let $\mathcal{T}$ be an optimal tour with total latency OPT. Let $\epsilon > 0$ be any parameter. Break this tour into k segments, so that in segment i we visit n_i nodes, where

$$\begin{aligned} n_i &= \left\lceil (1+\epsilon)^{k-1-i} \right\rceil \text{ for } i = 1 \ldots k-1 \\ n_k &= \lceil 1/\epsilon \rceil . \end{aligned}$$

Let the length of the ith segment be T_i. If we let $n_{>i}$ denote the total number of nodes visited in segments numbered $i+1$ and later, then a simple calculation shows that (and this was the reason for our choice of n_i's)

$$n_{>i} = \sum_{j>i} n_j \leq \frac{n_i}{\epsilon}, \text{ for every } i = 1 \ldots k-1 \qquad (1)$$

Now imagine doing the following in each segment except the last one: replace that segment by the minimum-cost traveling salesman path on the same subset of nodes, while maintaining the starting and ending points. We claim that the new latency is at most $(1+\epsilon)$OPT. First, note that $\sum_{j=1}^{m-1} T_j$ is a lowerbound on the latency of any node in the m'th segment. Adding over all segments, we get the following lowerbound on OPT:

$$\mathrm{OPT} \geq \sum_{i=1}^{k} n_{>i} \cdot T_i. \tag{2}$$

Consider the effect of replacing the ith segment with the shortest salesman path on that subset of nodes. The length of the segment cannot increase, and so neither can the latency of nodes in later segments. The latency of nodes within the segment can only rise by $n_i T_i$. Thus the new latency is at most the lowerbound in (2) plus

$$\sum_{i=1}^{k-1} n_i \cdot T_i. \tag{3}$$

Now condition (1) implies that the new latency is at most $(1+\epsilon)$OPT, as claimed.

Note that if, instead of the optimum salesman path in each segment, we use a $(1+\gamma)$-approximate salesman path, then the latency of the final tour is $(1+\gamma\cdot\epsilon+\gamma)$OPT.

2.1 Reduction from Minimum Latency to Weighted Vehicle Routing

The purpose of this section is to note that our technique described above implies a quasipolynomial-time approximation-preserving reduction from Minimum Latency to a version of Weighted Vehicle Routing. We do not know of a prior result along these lines.

In this version of weighted vehicle routing, we are given a set of n clients, who have to be visited by a fleet of m vehicles. Vehicle i has a designated depot s_i to start from and another depot t_i to finish at. It also has a *capacity* (which is the number of clients it can visit) c_i and a *per-mile cost* d_i. The goal is to assign clients to vehicles so as to respect the capacity constraints and minimize the total cost, which is the sum of distances covered by the vehicles, weighted by the per-mile cost.

Given a set of n nodes, the reduction proceeds as follows, where $k, n_i, n_{>i}$ have the same meaning as above: "Let p_0 denote the starting node of the minimum latency tour. For every sequence of k nodes $p_1, p_2, \ldots, p_k$, use the Vehicle Routing Oracle to construct a solution to the instance in which there are k vehicles, and the capacity of the ith vehicle is n_i, its per-mile cost is $n_{>i}$, and its starting and end depots are p_i and p_{i+1} respectively. At the end, output the lowest cost solution found (over the choice of all sequences of k points)."

Clearly, if the vehicle routing oracle computes a ρ-approximation in polynomial time, our reduction will lead to a $\rho(1+\epsilon)$-approximate solution in $n^{O(\log n/\epsilon)}$ time.

3 The Tree Case

The optimum salesman tour on a tree may in general need to visit a node unbounded number of times. For example in a star graph, it must visit the center node $n-1$ times. However, the tour never needs to visit an edge more than twice, as is easily check. Thus the optimum tour has very simple structure and can be found by depth-first search,

A minimum latency tour, on the other hand, could have a very complicated structure. Consider for example a complete binary tree in which all edges have zero weight except those attached to leaves. The minimum latency tour will first visit all the internal nodes in some arbitrary order, and then all the leaf nodes in sorted order by weight, thus crossing the root node $n/2$ times.

However, our technique from Section 2 allows us to view a near-optimum latency tour as a union of $O(\log n/\epsilon)$ salesman paths. By observing that within each salesman path an edge is only visited twice, we can then ensure that each edge is only visited $O(\log n/\epsilon)$ times overall. A similar idea underlies the proof of our *Structure Theorem* below.

First we define a recursive partition of the tree. We note that we can always find a node with the following property: Let $m > 3$ denote its degree and $f_1, \ldots, f_m$ denote the sizes of the components attached to the node. There exists a subset $S \subseteq \{1, \ldots, m\}$ such that

$$\lfloor \frac{n}{3} \rfloor \leq \sum_{i \in S} f_i \leq \lceil \frac{2n}{3} \rceil.$$

We designate this node as a *separator node* and the $|S|$ components as the *left* of the tree and the other $m-|S|$ as the *right* of the tree. (The node itself is copied twice and appears in both sides.) Then we recur on the two sides. This gives a recursive partition of the tree. We say that a tour *crosses* the separator node if it goes from the left side to the right.

As already noted, a minimum salesman tour is obtained by depth-first-search and therefore needs to cross each separator node only twice. The next theorem says that the near-optimal latency tour needs to cross each separator only $O(\log n/\epsilon)$ times.

Theorem 1 (Structure theorem for Weighted Trees) *The following is true for every integer $n > 0$ and every $\epsilon > 0$: For every well-rounded weighted tree on n nodes with a node-separator based partition as defined above, a tour exists with latency at most $(1+\epsilon)OPT$, that crosses each separator node only $O(\log n/\epsilon)$ times.*

Proof: Let $\mathcal{T}$ be the optimum tour. Divide it into $O(\log n/\epsilon)$ segments as in Section 2 and replace each segment by the optimum salesman path. Now use the fact that a minimum salesman path does not cross a separator node going from the left side to the right (or vice versa) of the partition at that node more than twice. □

A simple dynamic programming approach that relies on this structure can now be used to compute $(1+\epsilon)$-approximate latency tours for general trees as follows: we identify a separator node, "guess" the number of times the tour crosses this node, and for each crossing, the length of the tour portion after it and the number of nodes on that portion. Then we recur on each side of the separator node. Of course, by "guessing" we refer to exhaustive enumeration, which can be done in $n^{O(\log n/\epsilon)}$ time, since there are only $O(\log n/\epsilon)$ crossings in total. For each guess, we need to recursively compute a collection of paths visiting *all* the nodes on each side of the edge separator, where each path connects contains the guessed number of nodes and have the guessed path length. Thus the running time is $n^{O(\log n/\epsilon)}$.

4 The Euclidean Case

In Section 2, we reduced the minimum latency problem to the problem of finding a covering of the n nodes using $O(\log n/\epsilon)$ salesman paths. Now we use a simple modification of Arora's [2] Structure Theorem for Euclidean TSP and k-TSP to show that this set of $O(\log n/\epsilon)$ salesman paths together have a very simple structure. Thus they can be computed by dynamic programming in $n^{O(\log\log n/\epsilon)}$ time.

An instance is *well-rounded* if the minimum internode distance is 1 and the largest internode distance is $O(n^2)$. A dissection of the instance is a recursive partitioning into squares whereby each square is partitioned into four equal squares, and the partitioning stops when all squares have size 1. A *quadtree* is defined similarly except the partitioning stops as soon as the square has at most 1 node. A *randomly shifted dissection* is obtained by applying a random shift to the "middle line" of the dissection in both x and y directions, and then "wrapping around" the rest of the dissection. A *randomly shifted quadtree* is obtained from a randomly shifted dissection by stopping the dissection when a square (which could be "wrapped around") has at most 1 node in it. For details and figures of the above concepts, see [2]. On each corner of each dissection square, we place m portals, where m is a power of 2. (Thus a portal of a square is a portal in every descendent of the square.) An (m, k)-*light tour* is one that crosses each quadtree boundary at most k times, and always at a portal.

Theorem 2 (Structure theorem for Euclidean case) *There exist constants c, f such that the following is true for every integer $n > 0$ and every $\epsilon > 0$. For every well-rounded Euclidean instance with n nodes, a randomly-shifted dissection has with probability at least $1/2$ an associated tour that is $(c\log n/\epsilon, f\log n/\epsilon)$-light, and whose latency is at most $(1+\epsilon)OPT$, where OPT denotes the latency of the minimum latency tour. The tour crosses each portal at most $O(\log n/\epsilon)$ times.*

Proof: The main ideas are the same as in Arora's proof after we break up the minimum latency tour into $O(\log n/\epsilon)$ salesman paths. Another further observation is that Arora's proof relies on an expectation calculation, and we will use *linearity of expectations.*

Let $\mathcal{T}$ be the optimum tour. As described in Section 2, we break it up into $k = O(\log n/\epsilon)$ segments, where the ith segment has n_i nodes. We replace each segment by the optimum salesman path for that segment. Now we use Arora's proof of his structure theorem. He shows how a salesman tour/path can be modified so that it crosses the boundary of each dissection square at most $O(1/\epsilon)$ times, and always at one of the m portals. His main result is to show that this can be done in such a way for $m = O(\log n/\epsilon)$ so that the expected increase in the length of the tour/path is a multiplicative factor $(1+\epsilon/2)$. We apply his method simultaneously to our $O(\log n/\epsilon)$ salesman paths, thus ending up with a collection of paths which cross each portal at most $2k$ times (since a path never needs to cross a portal more than twice), and cross the boundary of each quadtree square at most $O(k/\epsilon)$ times.

As for the effect on the latency, note that we are interested in a *weighted* sum of path lengths, where the weight assigned to the ith salesman path is $n_i + n_{>i}$. Arora's analysis alongwith linearity of expectations implies that the expected increase in this weight is a multiplicative factor $(1 + \epsilon/2)$. We conclude that with probability at least $1/2$, the increase in latency is a factor at most $(1 + \epsilon)$. □

Thus it suffices to build a randomly shifted quadtree, and to find the optimum $(c\log n/\epsilon, f\log n/\epsilon)$-light tour for the values of f, c given by the Structure Theorem. (If we wish to increase the success probability from $1/2$ closer to 1, we merely repeat the algorithm a few times.) We describe how to find such a tour using dynamic programming as in [2]. For each square in the quadtree, we "guess" the portals where the tour enters the square, how many times it crosses each portal, and for each of these crossings, what is the length of the tour portion after that crossing, and how many nodes are on that tour portion. Note that guessing how many nodes are visited after each portal crossing also tells us the order in which they occur in the tour. We will call a portal a *follower* of another portal (in this guessed order) if it is visited immediately after the that portal.

Of course, by "guessing" we refer to exhaustive enumeration, which can be done in $n^{O(\log n/\epsilon^2)}$ time, since there are only $O(\log n/\epsilon^2)$ crossings in total. For each guess, we need to compute a collection of paths visiting *all* the nodes inside the square, where each path connects a portal and its follower, such that the path length and the number of nodes in it agrees with the difference between the corresponding guesses for the two portals. We maintain a dynamic programming table of size

$$\#\text{ quadtree squares} \times \#\text{ guesses per square}$$

which is at most $n^{O(\log n/\epsilon^2)}$. Building this table in a bottom-up fashion is easy.

5 The metric case

For the general metric case we give a simple 17.24-approximation algorithm that uses Garg's 3-approximation algorithm for k-TSP [6]. It has the same flavor as the approximation algorithms in [4, 7]. We note that [7] give an improved 10.78-approximation algorithm, which is more complicated.

The general approach is motivated by the observation in Section 2, and we use the numbers $n_1, n_2, \ldots, n_k$ defined there, where $k = O(\log n/\epsilon)$. We will choose $\epsilon = \sqrt{2}$.

The algorithm is as follows. Assume for simplicity that we know the last segment in the tour, which contains $\lceil 1/\sqrt{2}\rceil = 2$ vertices (i.e., 1 edge). Compute the rest of the tour as follows. Let P be the starting node. For $i = 1, \ldots, k$, find, using Garg's algorithm, a tour L_i that starts at P and visits n_i nodes that are not visited by $L_1, \ldots, L_{i-1}$. Of the two possible directions of each tour, pick the one that minimizes the latency. Output the concatenation of these tours (using shortcuts to avoid multiple visits to already visited points).

We analyze this algorithm as follows. Let $\mathcal{T}$ be the optimum tour. Denote by T_i the length of the i-th segment of $\mathcal{T}$ (which contains n_i nodes). Then, as in (2), a lowerbound on OPT, the latency of the optimum tour, is

$$\mathrm{OPT} \geq \sum_{i=1}^{k} n_{>i} \cdot T_i, \qquad (4)$$

where $n_{>i} = n_{i+1} + n_{i+2} + \cdots + n_k$ is the number of nodes appearing in the last $k - i$ segments.

Now let d_i be the length of the tour L_i computed by our algorithm. We claim that

$$d_i \leq 3 \times 2(T_1 + \ldots + T_i) \text{ for } i = 1, 2, \ldots, k-1 \qquad (5)$$

The reason for the "3" is that it is the approximation ratio of Garg's algorithm. The rest of the expression is explained as follows. Consider the closed tour starting from P, including the first i segments of $\mathcal{T}$, and returning to P. Its length is at most $2(T_1 + T_2 + \cdots + T_i)$, and it includes at least $(n_1 + n_2 + \ldots + n_i)$ nodes. At least n_i of these nodes are not in $T_1, \ldots, T_{i-1}$ (since these first $i-1$ segments contain in total only $n_1 + n_2 + \cdots + n_{i-1}$ nodes). Thus $2(T_1 + T_2 + \cdots + T_i)$ is an upperbound on the length of shortest tour that starts and finishes at P and includes at least n_i nodes not on $T_1, \ldots, T_{i-1}$. This finishes the justification for claim (5).

Now notice that the latency of the n_i new nodes visited during tour L_i is upperbounded by $\frac{1}{2} n_i d_i + n_i \sum_{l=1}^{i-1} d_l$. The second term is an upper bound for the total latency incurred due to the $i-1$ segments of $\mathcal{T}$ preceding the i-th segment. The first term is an upperbound for the latency due to the i-th segment itself, since the latency in the forward direction plus the latency in the backward direction is equal to $n_i d_i$, and we traverse tour L_i in the direction that minimizes the latency of the n_i new points visited. So the total latency of our solution —ignoring the last segment — is upperbounded by

$$\begin{aligned} A &= \sum_{i=1}^{k-1} \left[n_i \left(\frac{1}{2} d_i + \sum_{l=1}^{i-1} d_l \right) \right] \\ &= \frac{1}{2} \sum_{i=1}^{k-1} n_i d_i + \sum_{i=1}^{k-1} \sum_{l=1}^{i-1} n_i d_l \\ &\leq \frac{1}{2} \sum_{i=1}^{k-1} n_i d_i + \sum_{i=1}^{k-1} d_i n_{>i} \end{aligned}$$

From Section 2, $n_i \leq (1+\epsilon) n_{i+1}$ and $n_{>i} \leq \frac{n_i}{\epsilon}$, so

$$\begin{aligned} A &\leq \frac{1+\epsilon}{2} \sum_{i=1}^{k-1} n_{i+1} d_i + \frac{1+\epsilon}{\epsilon} \sum_{i=1}^{k-1} n_{i+1} d_i \\ &= \frac{(1+\epsilon)(2+\epsilon)}{2\epsilon} \sum_{i=1}^{k-1} n_{i+1} d_i \end{aligned}$$

Observing that $n_{>i} = \sum_{j=i+1}^{k} n_j$ and ignoring the last segment (which was computed optimally) we also have

$$\begin{aligned} \sum_{l=1}^{k-1} n_{l+1} d_l &\overset{(5)}{\leq} 6 \sum_{l=1}^{k-1} n_{l+1} \sum_{i=1}^{l} T_i \\ &= 6 \sum_{l=1}^{k-1} n_{>l} T_l \\ &\leq 6 \times OPT \end{aligned}$$

and thus

$$A \leq 3 \frac{(1+\epsilon)(2+\epsilon)}{\epsilon} OPT$$

The factor $\frac{(1+\epsilon)(2+\epsilon)}{\epsilon}$ is minimized for $\epsilon = \sqrt{2}$ and its value is $3 + 2\sqrt{2} = 5.828$. Thus $A \leq 17.24 OPT$.

acknowledgements

We thank Satish Rao for pointing us to a simplification of our original algorithm and Christos Papadimitriou for bringing [9] to our attention. Sanjeev Arora thanks Naveen Garg and Stefano Leonardi for introducing him to the minimum latency problem.

References

[1] F. Afrati, S. Cosmadakis, C. Papadimitriou, G. Papageorgiou, and N. Papakostantinou. The complexity of the traveling repairman problem. *Informatique Theorique et Applications*, **20**(1):79–87, 1986.

[2] S. Arora. Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. *Journal of the ACM* **45(5)** pp 1-30, Sep. 1998. Preliminary version in *Proceedings of 37th IEEE Symp. on Foundations of Computer Science(FOCS)*, pp 2-12, 1996.

[3] S. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy. Proof Verification and the Hardness of Approximation for Problems. *Journal of the ACM* **45(3)** pp 501-555, May 1998.

[4] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan and M. Sudan. The minimum latency problem. *Proceedings of 26th ACM Symp. on Theory Of Computing(STOC)*, pp 163–171, 1994.

[5] X. Deng and C. Papadimitriou. Exploring an unknown graph. *Proc. 31st IEEE Symp. on Foundations of Computer Science(FOCS)*, pp. 355-361, 1990.

[6] N. Garg. A 3-approximation for the minimum tree spanning k vertices. *Proc. 37th IEEE Symp. on Foundations of Computer Science(FOCS)*, pp.302–309, 1996.

[7] M. Goemans and J. Kleinberg. An improved approximation ratio for the minimum latency problem. *Proc. 7th ACM-SIAM Symposium on Discrete Algorithms(SODA)*, pp 152-158, 1996.

[8] M. Grötschel, L. Lovász, and A. Schrijver. Geometric Algorithms and Combinatorial Optimization. Springer Verlag, Berlin 1988.

[9] E. Koutsoupias, C. Papadimitriou and M. Yannakakis. Searching a fixed graph. *Lecture Notes in Computer Science(LNCS)* 1099, pp.280–289, Springer Verlag, 1996.

[10] J. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: Part II- A simple PTAS for the geometric k-MST, TSP and related problems. Preliminary manuscript, April 30, 1996. To appear in *SIAM J. Computing*.

[11] C. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, **18**(1):1–11, Feb. 1993.