Technical Report No. 2001-445

Locating The Median Of A Tree In Real Time*

Marius Nagy and Selim G. Akl
Department of Computing and Information Science
Queen's University
Kingston, Ontario K7L 3N6
Canada
Email: {marius,akl}@cs.queensu.ca

May 23, 2001

Abstract

Determining the optimal location of a switching center in a tree network of users is accurately modeled by the median problem. A real-time approach is used in this paper to investigate the dynamics of such a communication network in two cases: (1) a growing tree of nodes associated with equal demand rates, and (2) a stream of corrections that arbitrarily change the demand rates at the nodes.

The worst-case analysis performed in both situations clearly demonstrates the importance of parallelism in such real-time paradigms. It is shown that the error generated by the best sequential algorithm in the first case can be arbitrarily large. A synergistic behavior is revealed when the quality-up is investigated in the second case.

1 Introduction

Facility location problems have been intensively studied due to their practical applications. Determining the optimal location of an emergency medical service station in a predominately rural area consisting of several towns, or locating a distribution center to receive products from a factory and then distribute them to several markets are only two examples of such problems. In this paper, we focus on a particular topology, assuming that the network under consideration is a tree. This assumption is justified by the fact that tree is the minimal topology that ensures connectivity between any two nodes in the network.

*This research was supported by the Natural Sciences and Engineering Research Council of Canada.

The optimality criteria extensively considered in the location theory literature are the minimum distance sum criterion and the minimum eccentricity criterion [15, 17]. Consider an undirected and weighted tree $T = (V, E)$, where V and E are the vertex and edge sets of T, respectively. For any two distinct vertices $v, u \in V$ let $d(v, u)$ be the distance from v to u computed as the sum of the weights of the edges forming the unique path from v to u. Note that, because T is undirected, $d(v, u) = d(u, v)$. The distance sum from the vertices of T to a vertex $u \in T$ is $Sum_T(u) = \sum_{v \in V} d(v, u)$. The eccentricity from the vertices of T to the vertex $u \in T$ is $Ecc_T(u) = max_{v \in V} d(v, u)$.

While the minimum eccentricity criterion aims to minimize the *worst* possible behavior of a tree network, we are concerned in this paper with the other class of location problems, in which the objective function tries to optimize the *average* behavior. More precisely, we try to locate a vertex in a tree such that the sum of distances from all the other vertices to this vertex is the minimum possible. That special vertex is called the *median* of the tree and therefore this problem is referred to as the median problem or *minisum* location problem.

While an immediate illustration for the median problem is locating a single ambulance station in a network of towns, we formulate an application closer to the computer science area. Given a communication network with a tree topology, the problem asks for optimally locating a switching center in order to minimize the average transmission cost in the network. Although many factors can influence the transmission cost, we only take into consideration the distance between the nodes involved in a communication.

Formally, we are given a tree network T whose node set V is $\{v_1, v_2, \cdots, v_n\}$. Associated with each edge (v_i, v_j) of the tree is a weight $d(i, j)$ denoting the distance between v_i and v_j. In our problem, as in most minisum location problems, the network model is constructed so that demands for services occur only at the nodes. Thus with every node v_i we associate a demand rate $g(i), i = 1, 2, \cdots, n$. The units for the demand rates may be frequencies, such as number of calls per day, or probabilities, such as the probability of a call being generated over a one-hour period. In many facility location problems it is convenient for the demand rates to be normalized, such that $g(1) + g(2) + \cdots + g(n) = 1$. In such a case $g(i)$ can be interpreted as the conditional probability that the call was generated at v_i, given that there was a call.

Suppose that the switching center is located at a point x in the network, not necessarily a node. Then the average transmission cost from the switching center to a random node on the network is

$$Sum(x) = \sum_{i=1}^{n} g(i) d(x, i), \tag{1}$$

where $d(x, i)$ is the distance from $x \in T$ to $v_i \in T$ computed as the sum of the weights associated with the edges composing the unique path from x to v_i. If x coincides with v_i then $d(x, i) = 0$. In the case where x is located somewhere between two nodes v_i and v_j, we consider edge (v_i, v_j) as being partitioned into two edges (v_i, x) and (x, v_j), both with their corresponding weights.

Although a genuine average requires that the right-hand side of equation (1) be divided by n, we adopt the definition of $Sum(x)$ employed in the literature [15]. For the same reason, the terms *median* and *average* are both used in the context of the same problem, even though

they usually represent different concepts (particularly in statistics).

An *absolute median* is a location x^* that minimizes the average transmission cost $Sum(x)$. If we restrict the location of the median only to one of the nodes of the network then the median is called a *vertex median*. Hakimi [14] was the first to show that there exists at least one node in an undirected network which is an absolute median and introduced a simple algorithm for determining it. In our communication network problem we are also looking for a vertex median, that is, we want our switching center built at one of the network's nodes. Furthermore, we are interested in a real-time version of the problem, where at regular time-intervals a new node joins the network and the switching center might have to be relocated in order to ensure optimal performance.

A simple case, in which all the nodes in the network are associated with equal demand rates, is studied first. It is shown that even in this simple setup, the error generated during one step by the best possible sequential algorithm that deals with the real-time constraints imposed could be arbitrarily large. By comparison, the parallel algorithm developed to address the same situation manages to compute the optimal solution in due time, generating no error.

In the general case, where the demand rates associated with the nodes in the tree are arbitrary, the *quality-up* achieved by the parallel algorithm when compared with the best sequential effort is measured. Again, the improvement in the quality of the solution, brought about by parallelism, is arbitrarily large from the theoretical point of view. In fact, even under reasonably practical assumptions, the improvement in the average transmission cost is still superlinear in the number of processors used by the parallel algorithm.

The rest of the paper is organized as follows. Section 2 offers some preliminary results as well as a detailed description of the real-time environment into which the performances of the sequential and parallel algorithms are to be compared. The simple case, with equal demand rates, is analyzed in Section 3. The generalization to arbitrary demand rates is investigated in Section 4. Finally, the paper concludes with a short discussion in Section 5.

2 Preliminaries

Determining the location of a vertex median can be done efficiently, due to the tree topology of the underlying graph. Goldman's algorithm [13] solves this problem in $O(n)$ steps. In order to understand how this algorithm works we need to introduce the following notation. For any subset V' of the node set V, $g(V')$ denotes the sum of the demand rates associated with nodes in subset V', that is,

$$g(V') = \sum_{v_i \in V'} g(i), \ \textit{for any subset } V' \subseteq V.$$

The main result on which Goldman has built his algorithm is given in the following.

Proposition 1 *Node v_i is an absolute median of T if there exist subtrees $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ of T, where $V_1 \bigcap V_2 = \{v_i\}$, $V_1 \bigcup V_2 = V$, $E_1 \subseteq E$, $E_2 \subseteq E$, such that both the following conditions hold:*

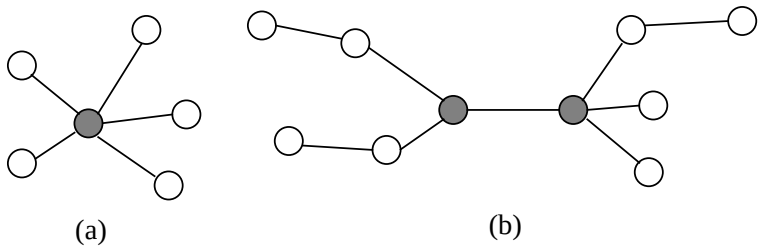

Figure 1: Example of a tree with (a) a unique vertex median; (b) two vertex medians.

(1) $g(V_1) \geq \frac{1}{2}g(V)$,

(2) $g(V_2) \geq \frac{1}{2}g(V)$.

This result might look surprising at first glance. The weights on the edges do not affect the location of the vertex median. Its position is solely determined by the demand rates at the nodes. A formal proof is given in [15] where it is shown that moving the median (in any direction) from a node v_i satisfying conditions (1) and (2) of Proposition 1 will not increase the average transmission cost. However, we may think of this result in a more intuitive way. An absolute median corresponds to that node of T minimizing the sum defined by equation (1). From the point of view of a combinatorial problem, a node has to be chosen out of n possible candidates, in such a way as to optimize a certain criterion. Changing the weight associated with an edge of T will modify accordingly the sums associated with all the nodes of T. Therefore, the process of selecting the node with the minimum sum is not influenced by such a modification. Different values for the weights on the edges of T lead to different values of the average transmission cost, but in all cases this sum is minimized by the same node(s).

The algorithm proposed by Goldman is based on tree contraction. It searches for an end node (a node with only one neighbor) v_i and its associated edge (v_i, v_j). It then modifies T by deleting v_i and (v_i, v_j) and increments $g(j)$ by $g(i)$. This procedure is repeated until an end node v_i is found with $g(i) \geq g(V)/2$, in which case that node is the desired absolute median. The case in which T is reduced to a single node is also possible, with the remaining node being the absolute median.

The efficiency of the algorithm is due to the fact that our distance measure on the tree is convex and thus at each iteration we are descending to a global minimum. This is the reason why even if the solution obtained by the algorithm does not satisfy both conditions of Proposition 1, that node is nevertheless an absolute median.

Note that the node v_i found by Goldman's algorithm is not necessarily the unique vertex median of the tree. The following observation settles this issue.

Observation 1 *A weighted tree T having demand rates associated with its nodes has at least one vertex median and at most two. In the case when there are two vertex medians, they are necessarily neighbors (Figure 1).*

To see that the above claim is true, suppose $T = (V, E)$ has two vertex medians VM_1 and VM_2. Consequently, there exists subtrees $T_1 = (V_1, E_1)$ rooted at VM_1 and $T_2 = (V_2, E_2)$

rooted at VM_2 of T such that $V_1 \bigcap V_2 = \emptyset$, $g(V_1) \geq \frac{1}{2}g(V)$ and $g(V_2) \geq \frac{1}{2}g(V)$. An easy way to determine such a subtree is to trace Goldman's algorithm and retain those deleted nodes leading to a vertex median. Since $V_1 \subseteq V$ and $V_2 \subseteq V$ it follows that necessarily $g(V_1) = g(V_2) = g(V)/2$. In other words, any node of T is included in one of the two disjoint sets V_1 and V_2. Therefore, there is no other node left that can claim the vertex median status or that can separate VM_1 and VM_2.

2.1 The real-time setup

The algorithm described above is able to determine the optimum location for the switching center in a static tree network of users. In reality, it is often the case that such a communication network is faced with the problem of an expansion. From time to time a new node has to be connected to the existing network while maintaining the average transmission cost to its minimum. In order to reach this goal, as soon as the current network is extended by connecting a new node, a decision has to be made as to whether or not another node in the network should assume the role of switching center. If the network is functioning more efficiently with the switching center placed at a new location, the node currently hosting the switching center should transfer its attributions in this respect to the newly chosen node. We refer to this process (of transferring the switching attributions) as *moving the switching center to its new location.* There are practical reasons for which a limited time is allowed when making the decision. The amount of time during which the network is functioning with a switching center not optimally located should be reduced as much as possible while allowing for the new center to be set up at its new location.

We can model the network expansion problem using the following real-time setup. Consider an initial tree T with n nodes. The location of a vertex median (VM) in T, corresponding to the switching center, is also given. Time is divided into equal intervals, each one measuring $c \log n$ time units, where c is a positive constant. At the beginning of each time interval a new node is connected to one of the nodes of T. Before the time interval ends, the network must be running with an optimally located switching center. In other words, the new vertex median must be found (and the switching attributions transfered) in at most $c \log n$ time units.

Locating the new vertex median before the imposed deadline is, evidently, the most challenging task. It is not difficult to set up the switching center at its new location in logarithmic sequential time, with respect to the number n of nodes served. One way to do this is to inform the involved nodes about their status change using their address in the network. Since there are n nodes in the whole network, $\log n$ sequential time suffices to decode the $\log n$-bit address of any of the network's nodes. Again, we will compare the performances of the best possible sequential algorithm and a parallel algorithm that deals with the median problem in this real-time environment.

3 A simple case: equal demand rates

Let us begin by analyzing the problem in the particular case when all the nodes in the network have equal demand rates. In this case, the demand rates at the nodes can be ignored

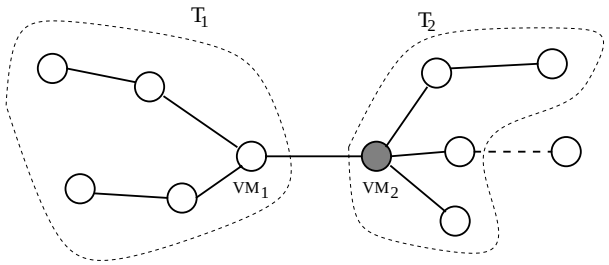

Figure 2: Adding a new node disqualifies one of the old vertex medians (VM_1).

altogether and the tree can be looked at as a usual weighted tree. The sufficient condition given in Proposition 1 for a node v_i to be an absolute median, can be simplified in this particular case as follows.

Proposition 2 *Node v_i is an absolute median of T if there exist subtrees $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ of T, where $V_1 \bigcap V_2 = \{v_i\}$, $V_1 \bigcup V_2 = V$, $E_1 \subseteq E$, $E_2 \subseteq E$, such that both the following conditions hold:*

(1) $|V_1| \geq \frac{1}{2}|V|$,

(2) $|V_2| \geq \frac{1}{2}|V|$.

How is the location of a vertex median affected by the addition of a new node to the tree? The following propositions summarize all the possibilities.

Proposition 3 *Adding a new node to a tree T with 2 vertex medians will disqualify one of them. The vertex median closer to the newly added node becomes the unique vertex median of the extended tree (Figure 2).*

Proof
Due to the fact that T has two vertex medians, the node set V of T can be partitioned into two subsets with equal cardinality: $|V_1| = |V_2| = |V|/2$ (note that T must have an even number of nodes). Each of these two subsets represents the node set of a subtree rooted at one of the two vertex medians. We already know from Observation 1 that VM_1 and VM_2 are neighbors. Suppose, without loss of generality, that the new node added to T is connected to one of the vertices of T_2, increasing the cardinality of V_2 to $1+(|V|/2)$. As it can easily be checked, VM_2 satisfies both conditions of Proposition 2 and is therefore a vertex median of the extended tree. On the other hand, VM_2 is the unique vertex with this property because the extra node added leads to an odd number of nodes in the extended tree, eliminating the possibility of having two vertex medians. □

Proposition 4 *The addition of a new node to a tree T with a unique vertex median will result in one of the following two situations:*

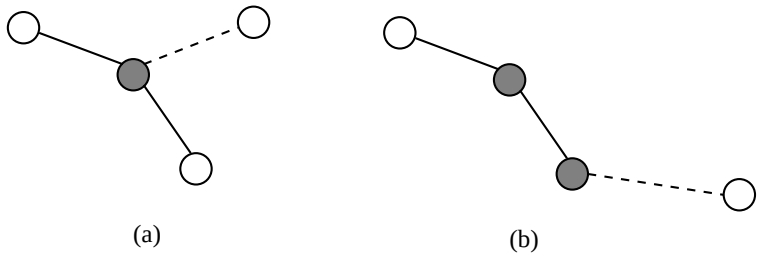

Figure 3: Addition of a node to a tree with a unique vertex median.

(a) the old vertex median remains the unique vertex median of the extended tree (Figure 3(a)).

(b) another node joins the old vertex median to become the second vertex median of the extended tree. The second median is the neighbor of the old median on the path to the newly added node (Figure 3(b)).

Proof

Suppose VM is the unique vertex median of the initial tree $T = (V, E)$ with n nodes. The sequence of nodes deleted by Goldman's algorithm in order to reach VM forms a tree $T' = (V', E')$ rooted at VM, such that $|V'| \geq n/2$. If $|V'| = n/2$ then it must be the case that v, the only neighbor of VM which is not in T', is the root of another tree $T" = (V", E")$ with $V"$ and V' being disjoint sets. $T"$ would account for the other half of the nodes of T, making v the second vertex median of T. But according to the hypothesis, VM is the only vertex median of T, so $|V'|$ must be strictly greater than $n/2$.

Because $|V'|$ must be an integer, it follows immediately that $|V'| \geq (n+1)/2$. If the new node is connected to T through a node outside T', then Goldman's algorithm will go through the same sequence of steps to find VM as a vertex median of the extended tree.

Alternatively, if the new node is attached to T', then VM can possibly lose its property only in favor of one of its children, say u. Let us denote by x the number of nodes in the subtree of T' rooted at u, before the addition of the new node. Then $x < n/2$ because u was not a vertex median of T, and in the same time $x + 1 \geq (n+1)/2$ represents the necessary condition for u to be a vertex median of the extended tree. The system formed by the last two inequalities is equivalent with

$$n - 1 \leq 2x < n.$$

The only integer x satisfying the above double inequality is

$$x = \frac{n-1}{2}.$$

This gives $x + 1 = (n+1)/2$, meaning that u and VM are each the root of a distinct tree with $(n+1)/2$ nodes. This corresponds to the case in which they will both be vertex medians, and u is that neighbor of VM whose corresponding subtree includes the newly added node. □

We can therefore conclude from these two propositions that adding a new node to a weighted tree with equal demand rates will either result in the old vertex median retaining

its property, or the adjacent vertex on the path from the old median to the new node becoming the new vertex median. We use these observations to design the sequential and parallel real-time algorithms for determining the location of the new vertex median.

3.1 Sequential algorithm

Given a weighted tree T with n nodes, a vertex median can be found in $O(n)$ time using Goldman's algorithm [13]. Considering T as a rooted tree with the vertex median as its root, and using a tree contraction procedure we can compute for each node v_i the number $l(v_i)$ of nodes in the subtree rooted at v_i. For this it is sufficient to associate the integer 1 with every node and make every non-leaf node compute the sum of its children and send the sum plus 1 to its parent. The whole computation will still take $O(n)$ time.

Having all this initial information, we want to determine the location of a vertex median when a new node is connected to T. To accomplish this we can use the basic idea of Goldman's algorithm. On the path from the most recently added node to the old vertex median, we increment the label associated with each node by 1, to account for the extra node that was added. As soon as we increment a node's value, we check it to see if it is equal to or larger than $(n+1)/2$, half of the total number of nodes in the extended tree. The first node to meet this criterion is declared the new vertex median. A formal description of the algorithm is given in the following.

Algorithm *Real_Time_Sequential_Vertex_Median*

Input: $\diamond$ The extended tree T' with $n+1$ nodes,
$\diamond$ OVM - the location of the old vertex median,
$\diamond$ $l(v_i), i = 1, \cdots, n$ - the labels associated with the nodes of the initial tree T.
Output: NVM - the location of the new vertex median.

1. Consider T' as a rooted tree with root OVM.
2. Let u = the parent of the newly added node in T'.
3. WHILE $u \neq$ OVM DO
   $l(u) = l(u) + 1$;
   IF $l(u) \geq \frac{n+1}{2}$ THEN
   NVM $= u$;
   Stop.
   ELSE
   let $u = parent(u)$;
4. NVM = OVM;
5. Stop.

The correctness of the above algorithm is easy to check, since it consists of a slightly modified version of Goldman's algorithm. The only difference is that here the nondetermin-

ism present in Goldman's algorithm is eliminated. We know exactly where to search for the new median, namely, on the path linking the old vertex median to the newly added node. According to Propositions 3 and 4, either the old median remains the unique vertex median of the extended tree (in which case Step 4 of the algorithm will be reached), or the neighbor of OVM on the path to the extra node will satisfy the condition for being a vertex median and consequently terminate the while loop of Step 3.

We conclude this section with the important observation that the above algorithm is the best possible sequential algorithm that deals with the worst case. It might be possible perhaps to construct an algorithm that is able to decide in constant time if the location of the vertex median remains unchanged or not. But in the worst case, when the median is shifted one step towards the newly added node, there is no way to tell which of the children of the old median is the new median, unless we reach it coming from the recently added node. Unfortunately, between the new node and the vertex median there could be on the order of n nodes, so the complexity of the sequential updating algorithm is still $O(n)$.

3.2 Parallel algorithm

In this section we develop a parallel algorithm capable of determining the precise location of a vertex median in the extended tree T'. The algorithm is presented in the following.

Algorithm *Real_Time_Parallel_Vertex_Median*

Input: $\diamond$ The extended tree T' with $n+1$ nodes,
$\diamond$ OVM - the location of the old vertex median.
Output: NVM - the location of the new vertex median.

1. Consider T' as a rooted tree with root OVM.
2. Assign the label 1 to each leaf and the function computing the sum of the children's labels to each internal node of T'. Use tree contraction to compute $l(v_i)$ for all nodes v_i of T', $i = 1, \cdots, n+1$.
3. Any node $u \neq$ OVM having $l(u) \geq \frac{n+1}{2}$ is marked as a vertex median (VM).
4. IF VM exists
THEN
IF $l(\mathsf{VM}) > \frac{n+1}{2}$ THEN NVM = VM;
ELSE NVM = OVM;
ELSE
NVM = OVM.

There are two observations worth formulating about the above algorithm. In the first place, note that there can only be one node meeting the requirements specified in Step 3. Secondly, note that the algorithm is constructed in such a way as to shift the location of the median only when it is absolutely necessary, that is, when the old median no longer satisfies

the required property.

The complexity of the algorithm is determined by the tree contraction step. Using a cost-optimal parallel tree contraction algorithm ([1, 12, 16]) the above computation can be completed in $c' \log n$ time units (where $c' > 0$), using $n/\log n$ EREW PRAM processors. The time needed for transferring the switching attributions from the old vertex median to the new vertex median is considered included in the $c' \log n$ time units. Recall that a new node is added to T at the beginning of each time interval, that is, every $c \log n$ time units. Assuming that $c' < c$, the parallel algorithm locates a new center successfully before a new node is received. Another advantage of using such a tree contraction procedure is that algorithm *Real_Time_Parallel_Vertex_Median* does not need any preliminary information to be computed during an initial preprocessing step.

3.3 Comparative analysis of the generated errors

We analyze the performances of the sequential and parallel updating algorithms described above by comparing the errors generated by them. Since we are dealing with an optimization problem, errors are measured as deviations of the computed solutions from the optimal solution. In our communication network problem, we are trying to minimize the *average transmission cost* (ATC), so the error produced by an algorithm $\mathcal{A}$ that tries to solve this problem is

$$error(\mathcal{A}) = computed\ ATC - minimum\ ATC.$$

In the worst case, the sequential algorithm does not have the resources to complete its computation and determine the location of the new vertex median in $c \log n$ time units. Alternatively, as it can be easily deduced from Propositions 3 and 4, moving the location of the vertex median is required only in the case when the old and the new vertex medians are the two vertex medians of the initial tree T, before the addition of the new node. In any other case, the current vertex median retains its property. It follows that a good heuristic for the sequential algorithm is to keep the location of the vertex median unchanged.

On the other hand, the parallel algorithm is always able to update the vertex median accurately, in any case. This means that the parallel algorithm always manages to compute the minimum ATC, producing the optimal solution and generating an error equal to 0.

In order to evaluate the sequential error caused by the addition of a new node, the worst possible case is depicted in Figure 4. Initially, the network has n nodes and 2 vertex medians, VM_1 and VM_2. Each of the two initial medians can be considered as the root of a subtree with $n/2$ nodes. The edge connecting VM_1 and VM_2 has a weight of W. At this moment, a new node u is attached to the subtree rooted at VM_2, thereby increasing its size to $n/2 + 1$. According to Proposition 3 the extended network will have only one vertex median, namely VM_2. The sequential algorithm will apply its heuristic and compute the average transmission cost considering that the switching center is located at VM_1. The error induced by this computation is:

$$error(Real\_Time\_Sequential\_Vertex\_Median) = Sum(VM_1) - Sum(VM_2) = W.$$

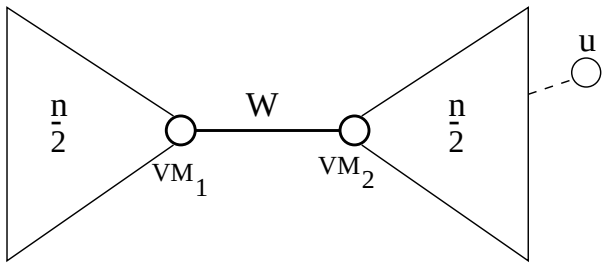

Figure 4: Worst case behavior of the vertex median during one step of the real-time computation.

It is interesting to note that this error could be arbitrarily large. For example, W could be an exponential function in n, making the error generated in one step by the sequential real-time algorithm exponential in the size of the network.

A high value for W will contribute not only to a large sequential error, but also to a large value for the optimal parallel solution. Therefore, asymptotically, the size of the sequential solution cannot grow beyond that of the parallel one. Nevertheless, this error translates into a highly increased average transmission cost in the sequential case, especially for large communication networks.

Furthermore, the error generated by the sequential algorithm continues to accumulate during the $O(n)$ possible time intervals for the real-time computation. This can make the price that has to be paid every time interval in the sequential case unacceptably high. Again, the parallel algorithm remains the first choice for such a real-time problem.

4 The general case: arbitrary demand rates

The simplifying assumption that all the nodes of the tree network have equal demand rates gives a certain stability to the vertex median location. In the worst case, all the added nodes shift the location of the vertex median in the same direction. Specifically, every two time intervals, the median moves only one node further away from its previous location.

In this section we analyze the more general case in which the demand rates at the nodes are arbitrary. We also focus on another kind of real-time paradigm. In the previous section, the real-time setup described corresponds to a *data-accumulating* paradigm ([10, 11]), where the size of the problem increases with every time interval.

The real-time algorithms constructed hereafter are *correcting* algorithms ([9]), that try to recompute the solution when one of the input data is altered. The set of input data for the problem of interest consists of the weights associated with each edge of the tree and the demand rates characterizing each node. The magnitude of the weights on the edges does not influence the location of the vertex median, but only the value of the average transmission cost. In this respect, the tree median problem can accommodate without any effort a real-time paradigm in which a stream of edge weight corrections is received by the communication network.

A more involved case, however, can be formulated when the corrections received in real time concern the demand rates at the nodes. Let us consider a weighted tree T with n

nodes, each node having associated a demand rate $g(i), i = 1, \cdots, n$. This tree models a communication network in which the frequency of calls generated by a certain node is taken into account through the demand rate associated with that node. For example, the value of $g(i)$ could represent the number of calls generated by node i during one day. For the purpose of computing the average transmission cost in such a network, the demand rates have to be normalized. In the following we make a distinction between the demand rate $g(i)$ associated with node i and its normalized value $p(i)$, computed as

$$p(i) = \frac{g(i)}{\sum_{i=1}^{n} g(i)}, \ i = 1, \cdots, n.$$

A normalized demand rate $p(i)$ represents the probability that the call was generated at node i, assuming that there was a call. The location of a vertex center of T with respect to the normalized demand rates $p(i), i = 1, \cdots, n$ is also given at the outset. This corresponds to the location of the switching center for the communication network.

For the real-time paradigm of interest in this section, a stream of demand rate corrections is received by the tree network. Specifically, every $c \log n$ time units (c being a positive constant) the frequency of calls generated by a certain node v_j changes. In other words, the value of $g(j)$ is modified. In order to adapt the network to the new situation, the location of the new vertex center of T must be found. The new vertex center corresponds to that node v_k for which

$$Sum(k) = \sum_{i=1}^{n} p(i)d(k, i)$$

is minimum. Being a real-time computation, a deadline is also imposed on when the result should be produced. Following the alteration of the demand rate of a certain node v_j, the switching center must be set up at its new location before a new correction arrives. Again, we assume that a switching center serving a network with n nodes can be easily set up in logarithmic sequential time using its $\log n$-bit address in the network.

4.1 Sequential solution

The modification of a single demand rate $g(j)$ for some j $(1 \leq j \leq n)$, will trigger the alteration of all the normalized demand rates $p(i)$, $i = 1, \cdots, n$. The computation of the new values for $p(i)$ will take $O(n)$ time when performed by a sequential computer. Having the normalized demand rates computed for each node, Goldman's algorithm can now be applied to determine the location of a vertex median.

Note that because the demand rates are arbitrary and the modifications can occur at any node, the location of the old vertex median is of no help in determining the location of the new vertex median. Indeed, the relative stability that characterizes the vertex median of a tree with equal demand rates disappears in the context of arbitrary demand rates. (Recall that in the simple case of equal demand rates, the new vertex median - when it does change its location - is one of the old median's neighbors.)

By contrast, when any of the tree's nodes can change its demand rate (which may increase as well as decrease) by an arbitrary amount, there are no procedures able to determine

efficiently which node will be the new median, using the location of the old median as a starting point. Practically, any node in the tree may become the new vertex median. Therefore, the nature of the problem imposes the computation of the new median location from scratch. Goldman's algorithm takes $O(n)$ time to find it.

It is worthwhile noting that we can obtain the actual value of the average transmission cost corresponding to the new median, without asymptotically increasing the running time of the algorithm. Considering T as a rooted tree with the new vertex median v_k as its root, a tree contraction scheme can be applied to compute $d(k,i)$ for all $i = 1, \cdots, n$. Finally, the sum $\sum_{i=1}^{n} p(i)d(k,i)$ can now be computed to produce the average transmission cost. Because each step requires $O(n)$ time, the whole computation (from receiving the modified demand rate $g(i)$ to obtaining the minimum average transmission cost) needs $O(n)$ running time for completion.

4.2 Parallel solution

Even if the demand rates at the nodes are arbitrary, the observation that T has at least one vertex median and at most two remains valid. It is also true that in the latter case, the two vertex medians are necessarily neighbors (Observation 1). These observations are important for designing the following parallel algorithm in charge of computing the minimum average transmission cost when one node v_j changes its demand rate $g(j)$.

Using $n/\log n$ EREW PRAM processors it is not difficult to recompute $p(i)$, for every $i = 1, \cdots, n$ in $O(\log n)$ time. Still in $O(\log n)$ time we can then compute the value

$$p(V_i) = \sum_{l \in V_i} p(l)$$

associated with each node v_i, where V_i is the set of nodes composing the subtree rooted at node v_i. This can be accomplished by considering T as a rooted tree and applying a cost-optimal parallel tree contraction algorithm. While the current vertex median can be selected as the root of T (as we did for equal demand rates), this decision is of no consequence for our parallel algorithm.

The node v_k with a minimum value for $p(V_k)$, but still greater or equal to $1/2$ is the desired new vertex median. Note the property that such a node v_k is unique. Finding it, as well as computing the corresponding average transmission cost can also be viewed as tree contractions applications. Consequently, the parallel real-time algorithm needs only $c'' \log n$ time units (where $c'' > 0$) for producing an optimal solution and setting up the switching center at the new location, using $n/\log n$ EREW PRAM processors. When $c" < c$, the parallel algorithm successfully computes the new vertex median before the end of the interval, that is, before the next change in the demand rate of a node occurs.

4.3 Quality-up analysis

The real-time nature of the computation discussed above has a major impact not on the speedup provided by the parallel algorithm, but on the quality-up achieved when parallelism is employed. Quality-up, an alternate measure for the performance of a parallel algorithm,

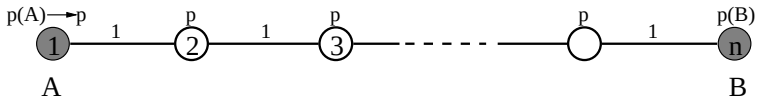

Figure 5: A possible setup yielding superlinear quality-up behavior.

is defined as the ratio of the quality of the parallel solution computed for some problem to the quality of the best sequential solution for the same problem. In the case of our specific problem, a solution is of good quality if the computed average transmission cost is low. Therefore, we define the quality of a computed solution as the inverse of the average transmission cost obtained.

Now that we have precisely defined what we want to evaluate, how large can the quality-up be, in the case of our communication network problem? In order to answer this question let us consider the case presented in Figure 5.

The n nodes are placed in such a way as to form an array-shaped network. For ease of presentation we make the following simplifying assumptions. Initially, all nodes have equal probabilities associated, except for the end nodes A and B. Without loss of generality, we also assume that all the edges have unitary weights associated. As we have already seen, the location of the vertex median is not affected by this assumption. The weights on the edges only affect the magnitude of the average transmission cost. Labeling the nodes with $1, 2, \cdots, n$ from left to right ($A = 1, B = n$), we can formally define the initial assumptions as follows:

(i) $p(i) = p$, for every $i = 2, \cdots, n-1$,
(ii) $p(B) >> p$,
(iii) $p(A) >> p(B)$,
(iv) $d(i, i+1) = 1$, for every $i = 1, \cdots, n-1$.

The second and third initial conditions will determine the vertex median to be located at node A. According to the real-time paradigm described for this problem, a modification occurs at one of the nodes, altering its associated probability. Consider the case in which the demand rate at node A decreases suddenly from $p(A)$ to p.

The new location of the vertex median has to be determined and the corresponding average transmission cost computed in no more than $c \log n$ time units. The sequential algorithm will obviously fail in trying to meet this deadline. In fact, the sequential machine will not even be able to compute the new normalized demand rates (probabilities) for all of the n nodes, following the modification incurred at node A. Lacking a better alternative, the sequential algorithm is forced to keep the switching center located at node A.

The parallel machine, on the other hand, has sufficient resources (processors) to complete the required computation in time and therefore obtain an optimal (minimum) solution. The ATC is computed by the parallel algorithm with respect to node B, the new vertex median of the tree:

$$Sum(B) = \sum_{i=1}^{n} p(i)d(B,i) = p\sum_{i=1}^{n-1} d(B,i) = (n-1)np/2.$$

If the network continues to function with its switching center located at node A, the average transmission cost incurred is

$$Sum(A) = \sum_{i=1}^{n} p(i)d(A,i) = p\sum_{i=2}^{n-1} d(A,i) + (n-1)p(B) = (n-2)(n-1)p/2 + (n-1)p(B).$$

Consequently, the quality-up is

$$quality - up = \frac{Sum(A)}{Sum(B)} = \frac{(n-2)p/2 + p(B)}{pn/2} = \frac{(n-2)p + 2p(B)}{pn}.$$

For definiteness and keeping in mind that initially $p(A) >> p(B) >> p$, we take $p(A) = O(1)$, $p(B) = O(n^{-\alpha})$ and $p = O(n^{-2\alpha})$, where $\alpha \geq 1$. Under these conditions, the improvement in the quality of the solution is on the order of $n^{\alpha-1}$.

Theoretically, this result proves that the improvement in the quality of the solution, gained by using a parallel algorithm, is unbounded, in the sense that α can be arbitrarily large. In practice, when α has some reasonable value (for example $\alpha = 2$), this result still demonstrates a superlinear improvement in the average transmission cost with respect to the number of processors employed.

5 Discussion

The paper addresses two important classes of real-time paradigms. The real-time setup defined for the equal demand rates case belongs to the data-accumulating paradigm. Any algorithm in charge of solving such a problem is faced with processing an increased amount of data at every time interval. On the other hand, the real-time environment where the demand rates are arbitrary belongs to the data-correcting paradigm. The size of the problem remains constant throughout the computation, but the algorithm in charge faces a new instance of the problem (even if not totally new) every time interval. To the same class belongs the paradigm in which a stream of edge-weight corrections is received by the communication network.

It might be worthwhile to study a third possible real-time arrangement, in which nodes may become disconnected from the network, thus decreasing the size of the problem. However, in this case, additional complications may arise. If the node to be removed is an internal node, then the removal operation will split the tree into several connected components. Still, if the removed node is only a leaf, then we believe that results similar to those obtained in this paper could be derived. The next step on this research path would be to investigate the performance achieved through parallelism in a real-time paradigm where any one of these cases is possible during one time interval.

From another perspective, the paper confirms that the classic approach in measuring the performance of a parallel algorithm (namely, by analyzing the speedup achieved) is not appropriate for real-time paradigms of the type defined herein (see also [4, 2, 3, 5, 6, 7, 8]). The true superiority of a parallel algorithm over its sequential counterpart, in the real-time area, can be fully exposed only if alternative performance measures are employed. These include the generated error and the quality-up, as defined in Sections 3 and 4, respectively.

It is also important to note that the synergy in the quality of the solution, proved for arbitrary demand rates, is even more remarkable if we realize that it was obtained for practical values of the parameter α, and is not just an abstract result that has no relationship with the real world.

References

[1] K. Abrahamson, N. Dadoun, D. G. Kirkpatrick, and T. Przytycka. A simple tree contraction algorithm. *Journal of Algorithms*, 10:287–302, 1989.

[2] S. G. Akl. Nonlinearity, maximization and parallel real-time computation. In *Proceedings of the Twelfth Conference on Parallel and Distributed Computing and Systems*, pages 31–36, Las Vegas, Nevada, November 2000.

[3] S. G. Akl. Parallel real-time computation: Sometimes quantity means quality. In *Proceedings of the International Symposium on Parallel Architectures*, pages 2–11, Dallas, Texas, December 2000.

[4] S. G. Akl. Superlinear performance in real-time parallel computation. Technical Report No. 2001-443, Department of Computing and Information Science, Queen's University, Kingston, Ontario, March 2001.

[5] S. G. Akl and S. D. Bruda. Improving a solution's quality through parallel processing. *The Journal of Supercomputing*, 19:219–231, 2001.

[6] S. G. Akl and S. D. Bruda. Parallel real-time optimization: Beyond speedup. *Parallel Processing Letters*, 9:499–509, 1999.

[7] S. G. Akl and S. D. Bruda. Parallel real-time cryptography: Beyond speedup II. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 1283–1289, Las Vegas, Nevada, June 2000.

[8] S. G. Akl and S. D. Bruda. Parallel real-time numerical computation: Beyond speedup III. *International Journal of Computers and their Applications, Special Issue on High Performance Computing Systems*, 7:31–38, 2000.

[9] S. D. Bruda and S. G. Akl. A case study in real-time parallel computation: Correcting algorithms. *Journal of Parallel and Distributed Computing*, 61:688–708, 2001.

[10] S. D. Bruda and S. G. Akl. On the data-accumulating paradigm. In *Proceedings of the Fourth International Conference on Computer Science and Informatics*, pages 150–153, Research Triangle Park, North Carolina, October 1998.

[11] S. D. Bruda and S. G. Akl. The characterization of data-accumulating algorithms. *Theory of Computing Systems*, 33:85–96, 2000.

[12] H. Gazit, G. L. Miller, and S.-H. Teng. Optimal tree contraction in an EREW model. In *Proceedings of the 1987 Princeton Workshop on Algorithm, Architecture and Technology Issues for Models of Concurrent Computation*, pages 139–156, Princeton, New Jersey, October 1987.

[13] A. J. Goldman. Optimal center location in simple networks. *Transportation Science*, 5:212–221, 1971.

[14] S. L. Hakimi. Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations Research*, 12:450–459, 1964.

[15] G. Y. Handler and P. Mirchandani. *Location on Networks: Theory and Algorithms.* MIT Press, Cambridge, Massachusetts, 1979.

[16] S. R. Kosaraju and A. L. Delcher. Optimal parallel evaluation of tree-structured computation by raking. In *Proceedings of the Third Aegean Workshop on Computing*, pages 101–110, Corfu, Greece, July 1988.

[17] B. C. Tansel, R. L. Francis, and T. J. Lowe. Location on networks: A survey. *Management Science*, 29:482–511, 1983.