DIMACS Technical Report 94-34
June 1994

Upper and Lower Bounds for Selection on the Mesh

by

Anne Condon [1]
Dept. of Computer Science
University of Wisconsin-Madison
Madison, WI 53706
condon@cs.wisc.edu

Lata Narayanan[2,3]
Dept. of Computer Science
Concordia University
Montreal, Canada H3G 1M8
lata@cs.concordia.ca

[1]Research supported by NSF grant numbers CCR-9100886 and CCR-9257241 and by awards from Digital Equipment Corporation and the AT&T Foundation.
[2]DIMACS Visiting Researcher
[3]Research supported in part by NSERC.

DIMACS is a cooperative project of Rutgers University, Princeton University, AT&T Bell Laboratories and Bellcore.

DIMACS is an NSF Science and Technology Center, funded under contract STC–91–19999; and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

A distance-optimal algorithm for selection on the mesh has proved to be elusive, although distance-optimal algorithms for the related problems of routing and sorting have recently been discovered. In this paper, we explain, using the notion of *adaptiveness*, why techniques used in the currently best selection algorithms cannot lead to a distance-optimal algorithm.

For worst-case inputs, we apply new techniques to improve the previous best upper bound of $1.22n$ of Kaklamanis *et al.* [7] to $1.15n$. This improvement is obtained in part by increasing the adaptiveness of previous algorithms. We also present the first algorithm for selection that has distance-optimal performance on average.

1 Introduction

A distance-optimal algorithm for selection on the mesh has proved to be elusive, although a series of algorithmic refinements has led to the development of distance-optimal algorithms for the related problems of routing and sorting. In this paper, we explain, using the notion of adaptiveness, why the techniques used in the best known algorithms for selection cannot lead to a distance-optimal algorithm. Furthermore, we develop an improved algorithm for median-finding on the mesh, in part by increasing the adaptiveness of previous algorithms for selection.

The selection problem is to find the element of some given rank k from $N = n^2$ totally ordered elements on a $n \times n$ mesh. Initially each processor contains exactly one element. The rank of an element x in a set S is $|\{y \in S \mid y \leq x\}|$. At the end of the algorithm the selected element must be at the center processor of the mesh. Because of the simplicity of its architecture, the mesh model is a well-studied and popular model of parallel computation [13] and is the basis for several parallel machines [1, 15]. We consider a standard model [6, 7, 13, 14, 17] in which a processor is allowed to communicate one packet of information to each of its neighbors during a single time step and to queue a constant number of packets between time steps.

Any selection algorithm requires $n - 1$ steps, since this is the distance from the corners to the center of the mesh. This is the only known lower bound for selection on the standard model of the mesh (Kunde's lower bound [11] applies only to a very restricted model of the mesh). The previously best algorithm for this problem runs in $1.22n$ steps [7, 9, 16]. Thus, the complexity of selection on the mesh is well-known to be $\Theta(n)$; the main open question is whether there exists a distance-optimal, or $n+o(n)$-time, algorithm for selection on the mesh. This question is especially interesting in light of the fact that distance-optimal algorithms for the related problems of sorting and routing have recently been disovered [6, 14, 9]. We provide a partial answer for this question in this paper.

Our main lower bound shows that the techniques used in the best previous selection algorithms cannot yield a distance-optimal algorithm. To explain this, we define a notion of adaptiveness for comparison-based algorithms on the mesh, and show that "weakly-adaptive" algorithms cannot be distance-optimal. Intuitively, median-finding algorithms adapt over time, based on new information learned from comparisons. For example, packets that appear likely to be the median, based on comparisons with a random sample of elements, may be routed towards the center early on. However, in all known median-finding algorithms, such adaptive routing decisions are made only once or twice. This is in part because gathering large random samples of elements is expensive.

To precisely limit the degree of adaptiveness of an algorithm, we limit the set of comparison results on which a processor's computation can depend. Roughly in our model, at fixed steps called knowledge steps, each processor learns the results of comparisons between every pair of elements that could possibly have reached the processor at that step. Between knowledge steps, processors may not perform new comparisons. However, at a step which is not a knowledge step, a processor may still learn new comparison results in the following

way: it learns the comparison results of its neighbors at the previous step. We say that an algorithm is *weakly-adaptive* if it has $O(1)$ knowledge steps; otherwise we say it is *highly-adaptive*. A *maximally adaptive* algorithm is one where every step is a knowledge step. We define this notion precisely in Section 2.2 and explain why the best previous algorithms for selection on the mesh are weakly adaptive.

In this paper, we show that there can be no distance-optimal weakly adaptive algorithm for selection on the mesh. We also show several other lower bounds for selection for highly adaptive algorithms that are restricted in other ways, such as in the number of packets that can be at one processor at any given time. Roughly, to prove our main result, we exploit the non-adaptiveness of the algorithm to show that there must be many possible candidates for the median at some time t. Since each processor can only store a constant number of packets, some candidate must be of distance much greater than $n - t$ from the center. Furthermore, on a possibly different input, this candidate is the true median, and is also sufficiently far from the median that it cannot reach the center in $(1 + \epsilon)n$ steps for some $\epsilon > 0$.

We next describe our upper bound results. We obtain the first algorithm for selection that has distance-optimal performance on average. We also improve the previous best upper bound for randomized algorithms, from $1.22n$ to $1.15n$. This upper bound is obtained in part by increasing the number of knowledge steps, and hence the adaptiveness, of the algorithm. Our new algorithm can be described as a "filtering" method: initially all elements are considered to be possible candidates for the median and are routed towards the center; then, over time, unlikely candidates are filtered out by a set of filtering processors to reduce the routing bottleneck close to the center. Processors that are equidistant from the center form a diamond-shaped filter at a given time. Each filtering processor uses sampling techniques to compute, at this time, a restricted range that is likely to contain the median; henceforth, elements routed to that processor which lie outside this range are discarded. Thus, the routes of packets are adapted at the filtering steps.

The success of the filtering method depends on the routing scheme, the locations of the filters and the times that filtering is done. The previous best algorithm of Kaklamanis *et al.* [7] can also be viewed as a filtering algorithm, but filtering is only done once, based on a single sample which is collected at the center. Our scheme uses three filters, and uses a new distributed sampling method to enable the filtering processors to filter elements earlier in the algorithm. A second contribution of our algorithm is in the routing scheme. In our algorithm, packets may be routed out of their quadrants, thus spreading the likely candidates for the median more uniformly over the mesh. This increases the effectiveness of the filtering method. We claim in our concluding section that our techniques push the filtering method to its limits, in that little further improvement to the running time can be obtained by increasing the number of filters.

To summarize, the main results of this paper are the following new upper and lower bounds for the median-finding problem on the standard, 2-dimensional mesh.

- We show that there is no weakly-adaptive distance-optimal algorithm for selection on the mesh which is distance-optimal. Our result holds even for randomized algorithms.
- We present a new $1.15n$ step randomized algorithm for selection on the mesh.
- We present a distance-optimal algorithm for selection on random inputs.

The rest of the paper is organized as follows. Our upper and lower bound models are defined and justified in Section 2. Our lower bounds are presented in Section 3, our upper bounds for worst-case and average-case inputs in Section 4. For simplicity, all our results are proved for the median-finding problem, but can be easily generalized for the general selection problem. Conclusions are presented in Section 6.

2 Upper and Lower Bound Models

2.1 Upper Bound Model

Our upper bound model is a standard one for describing routing algorithms on the mesh [6, 7, 14, 17]. The $n \times n$ mesh-connected array of processors (or two-dimensional mesh) contains $N = n^2$ processors arranged in a two-dimensional grid. More precisely, it corresponds to the graph, $G = (V, E)$, with the set of processors $V = \{(x, y) \mid x, y \in \langle n \rangle\}$ and the set of links, or edges $E = \{((x, y), (x, y+1)) \mid x \in \langle n \rangle, y \in \langle n-1 \rangle\} \cup \{((x, y), (x+1, y)) \mid x \in \langle n-1 \rangle, y \in \langle n \rangle\}$, where $\langle n \rangle$ denotes the set $\{1, \ldots, n\}$. In this paper, we will consider the case when n is odd; the case when n is even is treated similarly.

During a single parallel communication step, each processor can send and receive a single *packet* along each of its incident edges, where a packet consists of at most a single input element along with $O(\log N)$ bits of header information used for routing and counting purposes. Between communication steps, processors can store packets in their local queues, which are of bounded size (the *queue size*). Furthermore, a processor can perform a constant number of simple operations (such as copying, addition or comparison) on the elements and the header information of packets. In particular, a processor may *replicate* or create copies of a packet, and send different copies to different locations. The communication and computation of a processor may be a function of its internal memory, and, in the case of a randomized algorithm, of a constant number of coin flips. In measuring the performance of an algorithm, we count the number of communication steps, and assume that the bounded number of local operations that are performed by processors between communication steps are subsumed by the time needed for communication.

2.2 Lower Bound Model

We now explain and justify our lower bound model in detail. Our lower bound model applies to *comparison-based* algorithms for a class of problems where the input is a collection of packets, distributed among the processors of the mesh, and the output is just a reordering of the input. Such problems include sorting and selection. By restricting our model to comparison-based algorithms, we preclude the possibility, for instance, of processors routing packets differently based on whether they are odd or even valued. All previous algorithms for sorting and selection on the mesh are comparison-based algorithms.

To define a lower bound model which limits adaptiveness, we limit a processor's access to comparison results as follows. Fix an input I (if the algorithm is randomized, also fix the coin tosses of every processor). With each time t and processor p we associate a set of comparison results $S(p, I, t)$ as follows. Let $S(p, I, 0) = \emptyset$. In defining $S(p, I, t)$ for $t > 0$, we distinguish between two types of time steps. If t is a *knowledge step*, we let $S(p, I, t)$ be the set of comparison results between all pairs of packets that are *accessible to* p at time t. Here, an element Q is accessible to a processor p at time t if there is a path of length $\leq t$ from Q's initial location to processor p. If t is not a knowledge step, we let $S(p, I, t)$ be the union of $S(p, I, t-1)$ with the sets $S(p', I, t-1)$, where p' ranges over the processors adjacent to processor p in the mesh. Note that this does not imply that p has access to the results of comparisons between all pairs of elements accessible to it at time t. For example, suppose time step t is not a knowledge step, and A and B are packets with adjacent rank. Suppose furthermore that for any processor p', it is not the case that both A and B are accessible to p' at the most recent knowledge step. Then even if A and B are accessible to processor p at time t, $S(p, I, t)$ does not contain the result of the comparison between A and B.

With the definition of $S(p, I, t)$ in hand, we can now specify our lower bound model. As in our upper bound model, the queue size of each processor is bounded. We assume that the knowledge steps of the algorithm are a function of the input size n, say $g_1(n) < g_2(n) < \ldots < g_k(n)$. On a fixed input, at a given time step t, a processor p may do arbitrary internal computation, based on coin flips and the comparison results in $S(p, I, t)$. Based on this computation, a processor may send a packet along every edge incident to it. Finally, a processor may receive a packet along every edge incident to it.

We say that an algorithm is $k(n)$-*adaptive* if for all n, the algorithm has at most $k(n)$ knowledge steps. An algorithm is *weakly-adaptive* if it is $O(1)$-adaptive; otherwise we say it is *highly-adaptive*. An algorithm is *maximally adaptive* if every step of the algorithm is a knowledge step.

The maximally-adaptive model is similar to the lower bound model of Schnorr and Shamir [19], Kunde [11] and Han *et al.* [5]. However, our model allows replication of packets, which is not allowed in their model. On the other hand, their model is more general than our maximally adaptive model in that an algorithm may depend on the values of accessible packets, and not just on the outcomes of comparisons. However, we need the restriction to comparison-based algorithms only in proving lower bounds for maximally-adaptive algorithms also hold for algorithms which are not comparison-based. Thus, our lower bounds for maximally-adaptive algorithms are more general than previous results for such algorithms.

The previous best algorithm for selection of Kaklamanis *et al.* [7] is a weakly adaptive algorithm. Roughly, a small sample of elements is routed close to the center of the mesh and sorted. From this sample, splitters are computed which are broadcast to all processors within a certain range of the center. In our lower bound model, the splitters can be obtained "for free" in a single knowledge step. The routes of packets change only once during the course of the algorithm, based on the values of the splitters. Hence, the Kaklamanis *et al.* algorithm is actually a 1-adaptive algorithm. We will show that our algorithm of Section 4 is 3-adaptive.

In Section 3, we prove that there is no distance-optimal, weakly-adaptive algorithm for selection. Thus, if a distance-optimal algorithm for selection exists, the route of a packet must adapt, a number of times that grows with the size of the mesh, on the values of accessible elements at those times.

2.3 Notation

In discussing both upper and lower bounds, we use the following notation. The *center* processor is $(\lceil \frac{n}{2} \rceil, \lceil \frac{n}{2} \rceil)$. The *distance* between two processors (x_1, y_1) and (x_2, y_2) is $|x_1 - x_2| + |y_1 - y_2|$. For $r \leq n/2$, the *r-diamond* is the set of $2\lfloor r \rfloor^2 + 2\lfloor r \rfloor + 1$ processors that are at most r distant from the center processor. We consider the mesh to be composed of four *quadrants*, NW, NE, SW and SE. The NW quadrant contains exactly the processors in the set $\{(i,j) \mid 1 \leq i,j \leq \lceil n/2 \rceil\}$; the others are defined similarly. Since n is odd, the quadrants overlap at the borders and the center processor is in all quadrants. The restriction of the r-diamond to a given quadrant is called the *r-triangle* in that quadrant, and contains $(1/2)\lfloor r \rfloor(\lfloor r \rfloor + 1)$ processors. The *r-corner* of a quadrant is the set of processors in that quadrant of distance at least $n - r$ from the center. For example, in the NW quadrant, this is the set of processors $\{(i,j) \mid 1 \leq i + j \leq r\}$.

3 Lower Bounds

In this section, we present our main lower bound result, that there is no distance-optimal, weakly-adaptive algorithm for finding the median. Our lower bound holds even for randomized algorithms. We also prove lower bounds for restricted types of maximally adaptive algorithms.

3.1 Lower Bounds for Weakly Adaptive Algorithms

We prove our lower bound for a slightly more general model than that presented in Section 2.2. Let $S(I,t)$ be the union over all p of $S(p,I,t)$. In this new model, at every step, the computation of a processor may depend on $S(I,t)$ and not just $S(p,I,t)$. Intuitively, in this model, at each knowledge step a processor learns the comparison results accessible to *every* processor in the mesh. Between knowledge steps, a processor learns of no further

comparisons. Clearly this is more general than before, since $S(p, I, t) \subseteq S(I, t)$ for all t. Therefore, any lower bound on this model also applies to our model described in Section 2.2.

Our lower bound proof is somewhat simpler to present for this more general model. We need the following definition.

Definition 3.1 *Fix a deterministic algorithm and an input I. Element Q is* live *at time t if there is some input I' such that*
(i) Q is in the same initial position in I and I',
(ii) Q is the median of I' and
(iii) $S(I, t) = S(I', t)$.

In Lemma 3.1, we show that for a fixed, weakly-adaptive, deterministic algorithm and input, if some element Q is live at time $n - t$ and is far from the center at this time, then the algorithm cannot be distance optimal. In Theorem 3.1, we construct an input I which has many live elements at time $n - t$. We use the bound on the queue size to conclude that one of these must be far from the center at time $n - t$, and apply Lemma 3.1 to obtain the main result. In Theorem 3.2, we generalize this idea to randomized algorithms.

Lemma 3.1 *Fix a deterministic algorithm and an input I. Suppose that there is an $\epsilon > 0$ such that all packets containing some live element Q are of distance $\geq t + \epsilon n$ from the center, at some time $n - t$. Then the algorithm is not distance optimal.*

Proof: Since Q is live at time $n - t$, there is some input I' such that (i) Q is in the same initial position in I and I', (ii) Q is the median of I' and (iii) $S(I, n-t) = S(I', n-t)$. From (iii), it follows that in fact for all $t' \leq n - t$, $S(I, t') = S(I, t')$. Thus on inputs I and I', processors perform exactly the same operations at every step up to time $n - t$. Hence on inputs I and I', all packets containing Q are in the same positions at time $n - t$. Thus, on input I', the time to get Q to the center is at least $(n - t) + (t + \epsilon n) = (1 + \epsilon)n$. Since Q is the median of input I', the algorithm is not distance-optimal. □

We can now prove that there is no distance-optimal, weakly adaptive, deterministic algorithm for finding the median.

Theorem 3.1 *There is no distance-optimal, deterministic, weakly-adaptive algorithm for finding the median, for any constant queue size.*

Proof: Fix a weakly-adaptive algorithm which has k knowledge steps, say $n - f_1(n) < \ldots < n - f_k(n)$. Without loss of generality, we can assume that the first knowledge step is 0 and the last knowledge step is n; otherwise we can simply increase k by 2 and add two extra knowledge steps. Hence $f_1(n) = n$ and $f_k(n) = 0$. Let i be the smallest index such that $f_i(n) = o(n)$. Thus, on the fixed run, for some ϵ such that $0 < \epsilon n \leq n/2$ and any ϵ' such that $0 < \epsilon' < \epsilon$, (the value of ϵ' will be chosen later) there exists an infinite set of numbers M such that for all $n \in M$, $f_{i-1}(n) > \epsilon n$ but $f_i(n) + 1 < \epsilon' n$.

Fix $n \in M$ such that $\epsilon n > 2$. Let S be the set of elements in the $(\epsilon n - 1)$-corner of the NW quadrant. Let T be the set of elements in the $(\epsilon n - 1)$-corner of the SE quadrant. Note

that $(\epsilon n)^2/2 - O(n) \leq |S|$. Our goal is to construct an input I such that every element Q in S is live at time $n - f_i(n) - 1$. The input I' that will "witness" the fact that Q is live, will differ from I only in the set of input elements in T.

The input I is as follows, where all elements are integers. The elements in S are those elements $\leq$ median whose rank is within $|S| - 1$ of the rank of the median, and the elements in T are those elements greater than the median whose rank is within $|T|$ of the rank of the median. Also, the difference between each pair of elements in S is at least $|T|$.

We next show that at time $n - f_{i-1}(n)$, all elements of S are live. Since $S(I, n - f_{i-1}(n)) = S(I, n - f_i(n) - 1)$, it follows immediately that all elements of S are live at time $n - f_i(n) - 1$.

Let Q be any element of S. Suppose that the rank of Q is k less than the rank of the median, $0 \leq k \leq |S| - 1$. To show Q is live, we define I' to be the same as I, except that the k smallest elements in T are now chosen to be the elements just smaller than Q. Clearly, Q is in the same initial position in I and I', and Q is the median of I'. Hence properties (i) and (ii) of Definition 3.1 are satisfied.

It remains to show that $S(I, n - f_{i-1}(n)) = S(I', n - f_{i-1}(n))$. Suppose that (P, R) is any pair of elements in I (originating at processors p and p' respectively) and (P', R') is the corresponding pair of elements in I'. First suppose that it is not the case that that one element of each pair is in S and the other is in T. Then, by construction, the relative order of P and R is the same as the relative order of P' and R'. Thus, if $S(I, n - f_{i-1}(n))$ contains the result of the comparison between P and R, then $S(I', n - f_{i-1}(n))$ contains the same result between P' and R'. It remains to consider the case that one element of the pair is in S and the other is in T. We claim that in this case, in both the partial orders consistent with $S(I, n - f_{i-1}(n))$ and $S(I', n - f_{i-1}(n))$, the pairs (P, R) and (P', R'), respectively, are incomparable. From this it follows that $S(I, n - f_{i-1}(n)) = S(I', n - f_{i-1}(n))$.

We will show that in the partial orders consistent with $S(I, n - f_{i-1}(n))$ and $S(I', n - f_{i-1}(n))$, every element in S is incomparable with every element of T. This follows from two facts. The first is that on any input, at time $n - f_{i-1}(n) < n - \epsilon n$, it is not possible that both an element of S and an element of T are both accessible to the same processor. This is because the initial distance between any element in S and any element in T is at least $2n - 2\lfloor \epsilon n \rfloor + 1$; also any two elements can get at most a distance of 2 closer in one step. Thus, at any time $< n - \epsilon n$, an element in S and an element in T are of distance greater than $(2n - 2\lfloor \epsilon n \rfloor + 1) - (2n - 2\epsilon n) > 0$ from each other, and therefore cannot both reach the same processor. The second fact we use is that in both I and I', the elements in $S \cup T$ are exactly those elements whose ranks are in the range between $|S| - 1$ below the median and T above the median. Hence, if R is an element not in $S \cup T$, the elements in $S \cup T$ are all either $> R$ or $< R$, and so the relative order of an element of S and an element of T cannot be deduced from their order with respect to other elements. This concludes the proof that $S(I, n - f_{i-1}(n)) = S(I', n - f_{i-1}(n))$, and hence that Q is live at time $n - f_{i-1}(n))$.

Since the queue size is constant, we can choose some $\epsilon' > 0$ (depending on the queue size) such that all packets containing at least one of the elements Q of S are outside the $2\epsilon' n$-diamond at time $n - f_i(n) - 1$. Then from Lemma 3.1, it follows that the algorithm is not distance-optimal. □

We now extend our result to randomized algorithms. To prove a lower bound on the running time of a randomized algorithm, we need to show that there exists some input on which the expected running time is at least $(1+\epsilon)n$, for some $\epsilon > 0$. Thus, our technique of fixing an input and later modifying it (as in Theorem 3.1) is not valid for randomized algorithms. This is because it is not valid to modify the input *after* random bits of the algorithm are already chosen. Instead, we use a probabilistic argument. We show that there is a set of inputs, such that if one is chosen uniformly at random, then the expected running time (averaged now over both the uniform input distribution and the internal randomness of the algorithm) is $\geq (1+\epsilon)n$. We can therefore conclude that there exists an input on which the expected running time is at least $(1+\epsilon)n$. We note that this lower bound holds even if all processors have full knowledge of all coin tosses of all other processors.

Theorem 3.2 *There is no distance-optimal, randomized, weakly-adaptive algorithm for finding the median, for any constant queue size.*

Proof: Let the k knowledge steps, ϵ and M be defined as in Theorem 3.1. Consider the following set of inputs. Let S and T be the set of elements in the $(\epsilon n - 1)$-corner of the NW quadrant and the $(\epsilon n - 1)$-corner of the SE quadrant, respectively. The elements in the mesh outside of the set $S \cup T$ are chosen so that exactly half are greater than the elements of $S \cup T$ and exactly half are less than the elements of $S \cup T$. The elements of S are chosen to have values such that there is a difference of at least $|T|$ between any two members of S. The set T is chosen randomly in such a way that each of the elements of S is equally likely to be the median, and so that the relative order of the elements of T is the same for all random choices.

Fix any run of the algorithm, that is, fix the coin tosses of all processors. Consider any two inputs I and I' defined by different random choices of T. With respect to the fixed run, $S(I,t) = S(I',t)$ for all $t \leq n - f_i(n) - 1$. This can be proved in a manner similar to that used in Theorem 3.1 to show that $S(I, n - f_{i-1}(n)) = S(I', n - f_{i-1}(n))$. Hence, as in Lemma 3.1, with respect to the fixed run, the positions of all elements of S are the same on inputs I and I'. Choose ϵ' such that all packets containing at least $3/4$ of the elements of S are outside the $2\epsilon' n$-diamond at time $n - \epsilon' n < n - f_i(n) - 1$. The choice of ϵ' depends on the queue size but not on the input or the coin tosses.

Thus, the probability that the median is inside the $2\epsilon' n$-diamond at time $n - f_i(n) - 1$ is at most $1/4$ (where the probability is taken over the random choices of T since the run is fixed). If the median is inside the $2\epsilon' n$-diamond, the time to get it to the center is at least $n - \epsilon' n$. If the median is not inside the $2\epsilon n$-diamond, the time to get it to the center is at least $n + \epsilon' n$. Hence, the total expected time of the algorithm, with respect to the fixed run, is at least $1/4(n - \epsilon' n) + 3/4(n + \epsilon' n) \geq (1 + \epsilon'/2)n$. Since this is true for any fixed run, the total expected time, over all runs of the algorithm, also satisfies the same lower bound. □

3.2 Lower Bounds for Maximally Adaptive Algorithms

Using similar techniques, we can obtain lower bounds for maximally-adaptive algorithms, if we restrict the algorithms in various ways. In Theorem 3.3, we state this result. One way to restrict an algorithm is to restrict the queue size available at every edge. Another restriction is that packets may not be replicated (see the upper bound model, Section 2.1 for a definition). A third restriction is that packets must stay in the quadrant in which they are located initially; we say that packets are *quadrant constrained*. These restrictions are interesting because they again apply to all previously known algorithms for selection on the mesh. Our new algorithm in Section 4 is the first algorithm for selection which is not quadrant constrained. We note here that our lower bounds for maximally adaptive algorithms are proved on the model described in Section 2.2, not in the stronger version used in Section 3.1.

Parts (a) and (b) of Theorem 3.3 extend in different ways a previous result of Kunde [11], who proved a lower bound of $2n$ steps for maximally-adaptive algorithms with queue size of 1, and the additional restriction that packets are not replicated.

Our lower bounds for maximally adaptive algorithms apply to algorithms that are not restricted to be comparison-based. In other words, the action of a processor at time step t may depend on *values* of packets accessible to it, not just comparisons of packets accessible to it. This is similar to the models of Schnorr and Shamir and Kunde [19, 11]. We define a new notion of "liveness" of packets. The following definitions and lemmas pertain to a fixed maximally-adaptive algorithm. Given an element Q, let $Accessible(Q, I, t)$ be the union of the sets of input elements which are accessible to each packet containing element Q, on input I at the most recent knowledge step that occurs up to and including time t. We say an input I' is *consistent with* $Accessible(Q, I, t)$ if for each element $Q' \in Accessible(Q, I, t)$, element Q' is also in I', and is initially in the same location in both I and I'. We say element Q is *live at time* t if there is some input consistent with $Accessible(Q, I, t)$, such that Q is the median on that input. We next describe some useful facts about live elements, for a given algorithm and input I.

Fact 3.1 *Q is live at time t if $|Accessible(Q, I, t)| \leq n^2/2$.*

Fact 3.2 *Q is live at time t if Q is of rank at most k less than the median, and there is a set of at least k elements greater than the median which are not in $Accessible(Q, I, t)$.*

To see why Fact 3.2 is true, let I' be an input consistent with $Accessible(Q, I, t)$, obtained by changing the k elements greater than the median that are not in $Accessible(Q, I, t)$, so that they are now $< Q$. Then, Q is the median element of input I'. Hence by definition, Q is live at time t.

Lemma 3.2 *Fix an algorithm and an input I. If I' is consistent with $Accessible(Q, I, t)$ then*

$$Accessible(Q, I, t) = Accessible(Q, I', t).$$

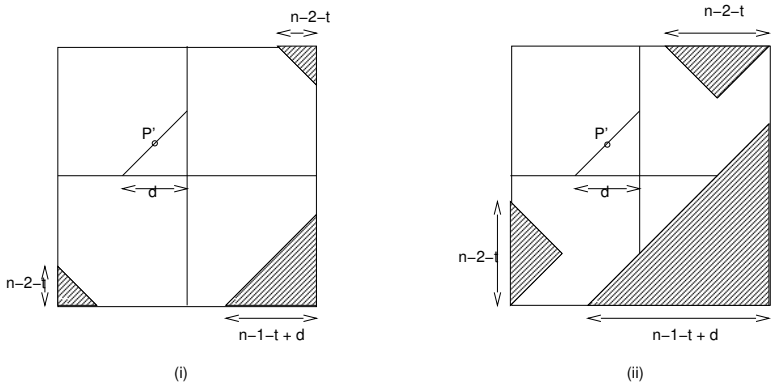

Figure 1: Packets in the shaded areas are inaccessible to processor P' at time t, where (i) $t - d < n/2$ and (ii) $t - d > n/2$.

Lemma 3.3 *Fix a deterministic algorithm and an input I. Suppose that there is an $\epsilon > 0$ such that all packets containing some live element Q are at distance $\geq t + \epsilon n$ from the center at some time $n - t$. Then the algorithm is not distance-optimal.*

Proof: Since Q is live at time $n-t$, there is some input I' consistent with $Accessible(Q, I, t)$ such that Q is the median element of input I'. On input I' at time $n - t$, all packets containing Q are also at distance $\geq t + \epsilon n$ from the center. This is because the positions of packets containing Q on inputs I and I' depend only on the input elements in $Accessible(Q, I, t)$, $Accessible(Q, I', t)$, respectively. Also, by Lemma 3.2, $Accessible(Q, I, t) = Accessible(Q, I', t)$. Hence, the time to get some packet containing Q to the center is at least $(1 + \epsilon)n$, and so the algorithm cannot be distance-optimal. □

The following claim is useful in proving part (b) of the theorem below.

Claim 3.1 *Suppose a packet P is outside the d-diamond at time $t \geq n/2$, where d is an integer. Then the number of elements not accessible to P at time t is at least*

(i) $(n - 2 - t)^2 + (1/2)(n - 1 - t + d)^2$, *if* $t - d > n/2$.

(ii) $(1/2)(n - 2 - t)^2 + (1/2)(n - 1 - t + d)^2$, *if* $t - d < n/2$,

Proof: The results are easily verified by referring to Figure 1. The shaded regions in these figures contain the locations of elements which are inaccessible to P', if P' in the NW quadrant, at distance d from the center, and is equidistant from the SW and NE quadrants. Also, the number of elements inaccessible to P' is a lower bound on the number of elements inaccessible to P. □

Theorem 3.3 *There is no distance-optimal, maximally adaptive deterministic algorithm for finding the median:*

(a) if the queue size is 1.

(b) if packets are not replicated and the queue size is 2.

(c) if packets are quadrant constrained and the queue size is 3.

(d) if packets are not replicated, are quadrant constrained and the queue size is 4.

Proof: Part (a): Fix any algorithm. The idea is to show that all elements are live at time $\lfloor n/2 \rfloor - 1$. Therefore, every element must be at some node. Since the queue size is 1, some live element is at a corner processor, and hence is at distance $n-1$ from the center processor at time $\lfloor n/2 \rfloor - 1$. Then it follows immediately from Lemma 3.3 that the algorithm requires $1.5n - 3$ steps.

We now show by induction on t that all elements are live at time $t \leq \lfloor n/2 \rfloor - 1$. Clearly this is true at time $t = 0$. Suppose it is true at time $t-1$. Then at the tth step, no element can be discarded, because at the tth step, the algorithm depends only on those elements accessible at time $t-1$. Hence at time t, every element must be at some node, and so there is exactly one packet containing every element, since the queue size is 1.

Recall that the elements accessible to any given element Q are exactly those elements initially within distance t from the processor containing Q at time t. Since $t \leq \lfloor n/2 \rfloor - 1$, at most $2t^2 + 2t + 1 \leq n^2/2$ elements are accessible to any given element Q. Hence by Fact 3.1, the element Q is live.

Part (b): Fix an input I. Just as in Theorem 3.3, since only one packet containing any element Q exists at any time, there are $\leq n^2/2$ elements in $\mathit{Accessible}(Q, I, t)$ at time $\lfloor n/2 \rfloor - 1$. Hence every element Q is live, by Fact 3.1.

Let $0 < \epsilon < 1/2$ (the value of ϵ will be chosen later). Consider the set S of live elements that are not inside the $(n/2 - \epsilon n)$-diamond at time $\lfloor n/2 \rfloor - 1$. There are at least $n^2 - 4(n/2 - \epsilon n)^2 - 4(n/2 - \epsilon n) - 2$ such elements, using the fact that the queue size is 2. At time $\lfloor n/2 \rfloor - 1 + \lfloor \epsilon n \rfloor$, these elements must still be outside the $(n/2 - 2\epsilon n)$-diamond.

At this time, applying Claim 3.1 part (i), for each element $P \in S$, there are still at least $(1/2)(\lfloor n/2 \rfloor - \lfloor \epsilon n \rfloor)^2 + (1/2)(n - 3\epsilon n)^2$ elements not accessible to Q, that is, not in $\mathit{Accessible}(Q, I, \lfloor n/2 \rfloor + \lfloor \epsilon n \rfloor - 1)$. Choose ϵ such that $(1/2)(\lfloor n/2 \rfloor - \lfloor \epsilon n \rfloor)^2 + (1/2)(n - 3\epsilon n)^2 \geq n^2/2$. Then, all elements in S are live at time $\lfloor n/2 \rfloor + \lfloor \epsilon n \rfloor - 1$, by Fact 3.1.

However, not all elements of S are inside the $(n/2 - \epsilon n)$-diamond. This is because at most $2\epsilon n(n/2 - \epsilon n)$ elements can cross into the $(n/2 - \epsilon n)$-diamond in $\lfloor \epsilon n \rfloor$ steps. Hence, at least $n^2 - 4(n/2 - \epsilon n)^2 - 4(n/2 - \epsilon n) - 2 - 2\epsilon n(n/2 - \epsilon n)$ elements of S are outside the $(n/2 - \epsilon n)$-diamond at time $\lfloor n/2 \rfloor + \lfloor \epsilon n \rfloor - 1$. This is $\Theta(n^2)$, for all ϵ in the range $0 < \epsilon < 1/2$. Hence for sufficiently large n, some element must be of distance at least $\lfloor n/2 \rfloor - \lfloor \epsilon n \rfloor + \epsilon' n$ from the center at time $\lfloor n/2 \rfloor + \lfloor \epsilon n \rfloor - 1$, for some $\epsilon' > 0$. By Lemma 3.3, the algorithm runs in time at least $(1 + \epsilon')n$.

Part (c): Let $t = \lfloor n/4 \rfloor$. Consider an input I in which the set S of all $2t^2$ elements just below the median are in a fixed quadrant, say NW, and exactly those elements greater than the median are outside the $n/2$-diamond.

Suppose that the queue size is at most 3. Then, at any given time, the number of distinct elements of S that are contained in packets inside the $(t + \epsilon n)$-triangle of the NW quadrant is at most $(3/2)(t + \epsilon n)(t + \epsilon n + 1)$. Choose ϵ such that $(3/2)(t + \epsilon n)(t + \epsilon n + 1) < 2t^2$. Then, at time $n - t - 1$, some element $P \in S$ is not contained in any packet inside the $(t + \epsilon n)$-triangle.

But at time $n - t - 1$, all elements initially in the $2t$-corner of the SE quadrant are not

accessible to any packet containing Q. There are at least $2t^2$ such elements. Thus by Fact 3.2, Q is live at time $n-t-1$. By Lemma 3.3, at least $(1+\epsilon)n-1$ time is required to find the median.

Part (d): Let $t = \lfloor n/4 \rfloor$ and consider an input I, just as in Part (c), in which the set S of all $2t^2$ elements just below the median are in a fixed quadrant, say NW, and exactly those elements greater than the median are outside the $n/2$-diamond.

We claim that all $2t^2$ elements in S are live at time $n-t-1$. This is because at least $2t^2$ elements outside the $(n/2+1)$-diamond are not accessible to every element of S (the set of $2t^2$ inaccessible elements may be different for different elements of S). Therefore, Fact 3.2 applies.

Suppose that the queue size is at most 4. Then, at time $n-t-1$, the set of distinct elements that are contained in packets inside the $(t-\epsilon n)$-triangle is of size at most $2(t-\epsilon n)(t-\epsilon n+1)$.

Hence, there is a set S' of at least $2t^2-2(t-\epsilon n)(t-\epsilon n+1)$ live elements that are not contained in packets inside the $(t-\epsilon n)$-triangle at time $n-t-1$.

All of the elements in S' are still live at time $n-t-1+\lfloor \epsilon n \rfloor$. This is because all are still outside the $(t-2\epsilon n)$-diamond. Therefore, at least $(t-1-\epsilon n)^2+(1/2)(2t-3\epsilon n)^2$ elements are not accessible, by Claim 3.1, part (ii). For $0 < \epsilon < 1/32$, $(t-1-\epsilon n)^2+(1/2)(2t-3\epsilon n)^2 \geq 2t^2$. Then, at least $2t^2$ elements greater than the median are not accessible to each element of S', and Fact 3.2 applies.

Also, at most $2\epsilon n(t-\epsilon n)$ elements of S' have crossed into the $(t-\epsilon n)$-diamond at time $n-t-1+\lfloor \epsilon n \rfloor$. Hence at least $2t^2-2(t-\epsilon n)(t-\epsilon n+1)-2\epsilon n(t-\epsilon n)$ live elements are still outside the $(t-\epsilon n)$-diamond at time $n-t-1+\lfloor \epsilon n \rfloor$. For any ϵ in the range $0 < \epsilon < 1/32$, $2t^2-2(t-\epsilon n)(t-\epsilon n+1)-2\epsilon n(t-\epsilon n) > 0$. Hence, one of these live elements is of distance at least $t-\epsilon n+\epsilon' n$ from the center at time $n-t-1+\lfloor \epsilon n \rfloor$, for some $\epsilon' > 0$. Hence by Lemma 3.3, the time required by the algorithm is at least $(1+\epsilon')n-2$. □

4 A Randomized Algorithm for Selection

The high-level description of our algorithm is similar to previous algorithms of [7, 10]. However, we use several new techniques, including greater adaptiveness, in order to obtain gains in the running time. We start this section with an outline of the general scheme used in previous algorithms, and then describe our new techniques in detail.

Briefly, the algorithm is as follows. All packets are routed inside a diamond, which we call the *gathering* diamond. At the same time, using sampling techniques, *bracketing elements* are computed by the center processor, and are broadcast to all processors in the gathering diamond. With high probability, the number of elements that lie between the bracketing elements is $O(N^{1-\delta})$ for some $\delta > 0$, and the median lies between the bracketing elements. All packets that lie between the bracketing elements are routed to a small block near the center. Since there are few of them, standard sorting algorithms can be used to sort them in $o(n)$ time. Once they are sorted and the exact ranks of the bracketing elements are

computed, the median can be identified and sent to the center. We summarize these steps in Figure 2.

It turns out that the main bottleneck to the running time of this algorithm is Step 1, that of routing all packets inside the gathering diamond. Note that the number of packets that can be routed inside the gathering diamond in one step is at most $8g$ where g (the gathering radius) is the distance from the center to the boundary of the gathering diamond. Thus at least $n^2/8g$ steps are required to complete this step. To overcome this bottleneck, our algorithm introduces several new techniques, which we now describe. The result is a more sophisticated Step 1, which is summarized in Figure 3 below.

1. All packets are routed inside gathering diamond.
2. The center processor selects bracketing elements $b < b'$ and broadcasts these to all processors in the gathering diamond. With high probability, there are $O(N^{1-\delta})$ input elements in the range $[b, b']$ and the median lies in this range.
3. All packets Q with $b \leq Q \leq b'$ are routed to a central block of side $o(n)$, and are sorted.
4. The ranks of b and b' are computed and are broadcast to all processors in the central block.
5. The element in the central block of rank $\lfloor N/2 \rfloor - rank(b)$ is the median. The processor with this element sends it to the center.

Figure 2: High-level Algorithm Description (Similar to Previous Algorithms).

First, our new algorithm filters out packets that are unlikely to be the median and routes only the remaining packets inside the gathering diamond. Since there are fewer packets to route, the value of g is reduced. Elimination or filtering of packets is done as follows. First, in Step 1a, all packets are routed inside the $n/8$-diamond. The processors on the boundary of the $n/8$-diamond are called the *filtering processors*. When Step 1a has been completed, a pair of *splitters* is selected by each filtering processor, such that the median is very likely to lie between the splitters. All packets that have been routed to filtering processor P and do not lie between P's splitters are thus eliminated.

Only the remaining packets, called the *live* packets, need to be forwarded inside the gathering diamond. Identification of the live elements and their routing are done in Steps 1d and 1e. Steps 1d and 1e are sped up by intermediate phases Steps 1b and 1c, which are overlapped with Step 1a and are completed before Step 1d begins. In Step 1b, a superset of the live packets, called the *tentatively live* packets, is identified. In Step 1c, approximately half of these are sent towards the gathering diamond. As a result, approximately half of the live packets at each filtering processor are already on their way to the gathering diamond by Step 1d, and so only half need to be sent during Step 1e. In Step 1b, the tentatively live packets are identified by computing yet another pair of splitters.

We note that our use of splitters to eliminate packets in Steps 1c and 1e are natural approximations of knowledge steps. Information is collected at selected processors to enable

1. A set of packets, which contains the median with high probability, is routed inside the gathering diamond, as follows.

   (a) All packets are routed inside the $n/8$-diamond. For each packet Q, either Q is routed inside the gathering diamond, or it is routed to a filtering processor P (i.e. on the $n/8$-diamond), chosen randomly and uniformly from $3n/8$ filtering processors.

   (b) Each filtering processor P selects a pair of splitters, $s_1(P) < s'_1(P)$, at time $n - 0.03n + o(n)$. The elements routed to P in Step 1a, which also lie in the range $[s_1(P), s'_1(P)]$, are called P's tentatively live elements.

   (c) At time $n-0.03n+o(n)$, each filtering processor selects, from its tentatively live packets, a random subset of size $0.03n$. The selected packets are routed towards the gathering diamond by time $n + o(n)$.

   (d) Each filtering processor P selects a pair of splitters, $s_2(P) < s'_2(P)$, at time $n+o(n)$. The elements routed to P in Step 1a, which also lie in the range $[s_1(P), s'_1(P)] \cap [s_2(P), s'_2(P)]$, are called P's live elements.

   (e) All live elements remaining at the filtering processors are routed inside the gathering diamond.

Figure 3: New Algorithm, Step 1.

them to *adapt* the routes of the packets that reside at them. Our overall algorithm is therefore consistent with the weakly-adaptive model with 3 knowledge steps, one at time $n - 0.03n$ in Step 1c, the second at time n in Step 1e, and the last one in Step 5. In contrast, previous algorithms used only one knowledge step, in Step 5. Thus our algorithm is more adaptive than previous algorithms.

One further idea is used to minimize the number of live packets at every processor. In Step 1 of previous algorithms, each packet Q is routed inside the gathering diamond, via a processor P on the boundary of the gathering diamond, where P is in the same quadrant as Q's initial location. In contrast, for a given packet, Step 1a of our new algorithm randomly selects a filtering processor from three quadrants, as the possible destination of the packet. The resulting routing scheme is quite complex, but we will see that as a result, packets that are likely to be the median are distributed over a larger number of processors, and hence, each processor has fewer live packets.

These techniques combined yield a significant improvement to Step 1. The remaining steps are just as before, although the implementation of Step 4 is slightly different than in previous algorithms.

This algorithm successfully halts in time $1.15n$ with high probability; by this we mean with probability $1 - O(N^{-\beta})$ for $\beta = 3$ (a larger value of β is possible simply by changing some parameters in the algorithm) . If the algorithm fails to route the median to the center processor in the required time, then the center processor broadcasts a message to restart, and a naive $O(n)$ algorithm can be executed (such as an algorithm based on sorting [19]).

Since the probability of failure is $O(N^{-3})$, the expected running time of the whole algorithm is still $1.15n$.

It is now possible to give an overview of the running time of the algorithm. First, consider Step 2. The bracketing elements are selected by the center processor using sampling techniques; to get a sample of the entire input to the center requires n steps since this is the maximum distance of an element to the center. Broadcasting the bracketing elements similarly requires time equal to g, the distance from the center to the processors on the boundary of the gathering diamond. Thus, Step 2 requires time $n + g$. Similarly, Step 3 requires time equal to $g - o(n)$, since elements have to be routed in from the processors on the boundary of the gathering diamond to the central block. It turns out that Step 4 can be overlapped with Step 3; and Step 5 requires only $o(n)$ time, since it involves computation in a small central block. Thus, the total time of Steps 2 through 5 must be at least $n + 2g$. In fact, they can be completed in time $n + 2g + o(n)$ as shown in [7].

It remains to consider Step 1. We summarize the analysis here; the details are presented in the full paper. Since at most n packets can be routed inside the $n/8$-diamond in one