

# Distributed Selection: A Missing Piece of Data Aggregation

By Fabian Kuhn, Thomas Locher, and Roger Wattenhofer

## Abstract

**In this article, we study the problem of distributed selection from a theoretical point of view. Given a general connected graph of diameter  $D$  consisting of  $n$  nodes in which each node holds a numeric element, the goal of a  $k$ -selection algorithm is to determine the  $k^{\text{th}}$  smallest of these elements. We prove that distributed selection indeed requires more work than other aggregation functions such as, e.g., the computation of the average or the maximum of all elements. On the other hand, we show that the  $k^{\text{th}}$  smallest element can be computed efficiently by providing both a randomized and a deterministic  $k$ -selection algorithm, dispelling the misconception that solving distributed selection through in-network aggregation is infeasible.**

## 1. INTRODUCTION

There is a recent growing interest in distributed aggregation, thanks to emerging application areas such as, e.g., data mining or sensor networks.<sup>2,8,23,24</sup> The goal of distributed aggregation is to compute an aggregation function on a set of distributed values, each value stored at a node in a network. Typical aggregation functions are *max*, *sum*, *count*, *average*, *median*, *variance*,  $k^{\text{th}}$  *smallest*, or *largest value*, or combinations thereof such as, e.g., “What is the average of the 10% largest values?”

The database community classifies aggregation functions into three categories: distributive (*max*, *min*, *sum*, *count*), algebraic (*plus*, *minus*, *average*, *variance*), and holistic (*median*,  $k^{\text{th}}$  *smallest*, or *largest value*). Combinations of these functions are believed to support a wide range of reasonable aggregation queries.\*

It is well known that distributive and algebraic functions can easily be computed using the so-called *convergecast* operation executed on a pre-computed *breadth first search* (BFS) tree: The root of the tree floods a message to the leaves of the tree, asking the leaves to start the aggregation. The inner nodes of the spanning tree wait until they have received the aggregated data from all their children, apply the aggregation function to their own data and the aggregated data, and subsequently forward the aggregation result to their respective parent. Convergecast is fast, as it terminates after at most  $2D_T$  time, where  $D_T$  denotes the depth of the spanning tree. Note that the depth of a BFS tree is at most the diameter  $D$  of the original graph  $G$ , thus a single convergecast costs merely  $2D$  time. An example for such a spanning tree in the context of sensor networks is depicted in Figure 1. However,

it is believed that holistic functions cannot be supported by convergecast. After all, the very name “holistic” indicates that one “cannot look into” the set of values, more precisely, that all the values need to be centralized at one node in order to compute the holistic function. Bluntly, in-network aggregation is considered to be practically impossible for holistic functions.

For arbitrary  $k$ , a selection algorithm answers questions about the  $k^{\text{th}}$  smallest value in a set or network. The special case of the  $k$ -selection problem where  $k = n/2$  is the well-known *median problem*. Generally speaking, selection solves aggregation queries about order statistics and percentiles. Surprisingly, little is known about distributed (network) selection, although it is critical to the understanding of data aggregation.

In this article, we shed some new light on the problem of distributed selection for general networks with  $n$  nodes and diameter  $D$ . In particular, we prove that distributed selection is strictly harder than convergecast by giving a lower bound of  $\Omega(D \log_p n)$  on the time complexity in Section 5. In other words, to the best of our knowledge, we are the first to formally confirm the preconception about holistic functions being strictly more difficult than distributive or algebraic functions. In addition, in Section 4.1, we present a novel *Las Vegas algorithm* which matches this lower bound with high probability, improving the best randomized algorithm. As for many networks this running time is strictly below collecting all values at one node, our new upper bound proves that (contrary to common belief) in-network aggregation is possible also for holistic functions; in fact, in network topologies where the diameter is large, e.g., in grids or in typical wireless sensor networks, selection can be performed within the same asymptotic time bounds as convergecast. As a third result, in Section 4.2, we derandomize our algorithm and arrive at a deterministic distributed selection algorithm with a time complexity of  $O(D \log_p^2 n)$  which constitutes a substantial improvement over prior art.

## 2. RELATED WORK

Finding the  $k^{\text{th}}$  smallest value among a set of  $n$  elements is a classic problem which has been extensively studied in the past approximately 30 years, both in distributed and non-distributed settings. The problem of finding the median, i.e., the element for which half of all elements are smaller and the other half is larger, is a special case of the  $k$ -selection problem which has also received a lot of attention. Blum et al.<sup>1</sup> proposed the first deterministic sequential algorithm that, given an array of size  $n$ , computes the  $k^{\text{th}}$  smallest element in  $O(n)$  time. The algorithm partitions the  $n$  elements into roughly  $n/5$  groups of five elements and determines the

\*We encourage the reader to think of a natural aggregation (single value result) query that cannot be formulated by a combination of distributive, algebraic, and holistic functions.

**Figure 1: Data in sensor networks is aggregated on a pre-computed virtual spanning tree. Typically, nodes receive data from their children and forward the aggregation result to their parent in the tree.**



median element of each group. The median of these  $n/5$  medians is then computed recursively. While this median of medians is not necessarily the median among all  $n$  elements, it still partitions all elements well enough in that at least (roughly) 30% of all elements are smaller, and also at least 30% are larger. Thus, at least 30% of all elements can be excluded and the algorithm can be applied recursively to the remaining elements. A careful analysis of this algorithm reveals that only  $O(n)$  operations are required in total. Subsequently, Schönhage et al.<sup>19</sup> developed an algorithm requiring fewer comparisons in the worst-case.

As far as distributed  $k$ -selection is concerned, a rich collection of algorithms has been amassed for various models over the years. A lot of work focused on special graphs such as stars and complete graphs.<sup>9,16</sup> The small graph consisting of two connected nodes where each node knows half of all  $n$  elements has also been studied and algorithms with a time complexity of  $O(\log n)$  have been presented.<sup>3,15</sup> For deterministic algorithms in a restricted model, this result has been shown to be tight.<sup>15</sup> Frederickson<sup>14</sup> proposed algorithms for rings, meshes, and also complete binary trees whose time complexities are  $O(n)$ ,  $O(\sqrt{n})$ , and  $O(\log^3 n)$ , respectively.

Several algorithms, both of deterministic<sup>12,13,20</sup> and probabilistic nature,<sup>17,18,20</sup> have also been devised for arbitrary connected graphs. Some of these deterministic algorithms restrict the elements the nodes can hold in that the maximum numeric item  $x_{max}$  has to be bounded by  $O(n^{O(1)})$ . Given this constraint, applying binary search results in a time complexity of  $O(D \log x_{max}) = O(D \log n)$ .<sup>13</sup> Alternatively, by exponentially increasing the initial guess of  $x_k = 1$ , the solution can be found in  $O(D \log x_k)$ .<sup>12</sup> To the best of our knowledge, the only non-restrictive deterministic  $k$ -selection algorithm for general graphs with a sublinear time complexity in the number of nodes is due to Shrira et al.<sup>20</sup> Their adaptation of the classic sequential algorithm by Blum et al. for a distributed setting has a worst-case running time of  $O(Dn^{0.9114})$ . In the same work, a randomized algorithm for general graphs is presented. The algorithm simply inquires a random node for its element and uses this guess to narrow down the number of potential elements. The expected time complexity is shown to be  $O(D \log n)$ . Kempe et al.<sup>7</sup> proposed a gossip-based algorithm that, with probability at least  $1 - \epsilon$ , computes the  $k^{\text{th}}$

smallest element within  $O((\log n + \log \frac{1}{\epsilon}) + (\log n + \log \log \frac{1}{\epsilon}))$  rounds of communication on a complete graph.

If the number of elements  $N$  is much larger than the number of nodes, in  $O(D \log \log \min\{k, N - k + 1\})$  expected time, the problem can be reduced to the problem where each node has exactly one element using the algorithm proposed by Santoro et al.<sup>17,18</sup> However, their algorithm depends on a particular distribution of the elements on the nodes. Patt-Shamir<sup>13</sup> showed that the median can be approximated very efficiently, again subject to the constraint that the maximum element must be bounded by a polynomial in  $n$ .

### 3. MODEL AND DEFINITIONS

In our system model, we are given a connected graph  $G = (V, E)$  of diameter  $D$  with node set  $V$  and edge set  $E$ . The cardinality of the node set is  $|V| = n$  and the nodes are denoted  $v_1, \dots, v_n$ . The *diameter* of a graph is the length of the longest shortest path between any two nodes. Each node  $v_i$  holds a single element  $x_i$ .<sup>†</sup> Without loss of generality, we can assume that all elements  $x_i$  are unique. If two elements  $x_i$  and  $x_j$  were equal, node IDs, e.g.,  $i$  and  $j$ , could be used as tiebreakers. The goal is to efficiently compute the  $k^{\text{th}}$  smallest element among all elements  $x_1, \dots, x_n$ , and the nodes can achieve this goal by exchanging messages. Nodes  $v_i$  and  $v_j$  can directly communicate if  $(v_i, v_j) \in E$ .

The standard asynchronous model of communication is used. Throughout this article, the communication is considered to be reliable, there is no node failure, and all nodes obediently follow the mandated protocol. We do not impose any constraint on the magnitude of the stored elements. However, we restrict the size of any single message such that it can contain solely a constant number of both node IDs and elements, and also at most  $O(\log n)$  arbitrary additional bits. By restricting the size of the messages, we strive to capture how much *information* has to be exchanged between the nodes in order to solve the problem. Moreover, such a restriction is quite natural as the message size is typically

<sup>†</sup> Our results can easily be generalized to the case where more than one element is stored at each node. The time complexities are then stated in terms of the number of elements  $N > n$  instead of the number of nodes.

limited in practical applications. Note that without this restriction on the message size, a single convergecast would suffice to accumulate all elements at a single node, which could subsequently solve the problem locally.

As both proposed algorithms are *iterative* in that they continuously reduce the set of possible solutions, we need to distinguish between nodes holding elements that are still of interest from the other nodes. Henceforth, the first set of nodes is referred to as *candidate nodes* or *candidates*. We call the reduction of the search space by a certain factor a *phase* of the algorithm. The number of candidate nodes in phase  $i$  is denoted  $n^{(i)}$ .

We assume that all nodes know the diameter  $D$  of the graph. Furthermore, it is assumed that a BFS spanning tree rooted at the node initiating the algorithm has been computed beforehand. These assumptions are not critical as both the diameter and the spanning tree can be computed in  $2D$  time.<sup>14</sup>

The main complexity measure used is the time complexity which is, for deterministic algorithms, the time required from the start of an execution to its completion in the worst-case for every legal input and every execution scenario. The time complexity is normalized in that the slowest message is assumed to reach its target after one time unit. As far as our randomized algorithm is concerned, we determine the time after which the execution of the algorithm has completed with high probability, i.e., with probability at least  $1 - \frac{1}{\pi^c}$  for a constant  $c \geq 1$ . Thus, in both cases, we do not assign any cost to local computation.

#### 4. ALGORITHMS

The algorithms presented in this article operate in sequential phases in which the space of candidates is steadily reduced. This pattern is quite natural for  $k$ -selection and used in all other proposed algorithms including the non-distributed case. The best known deterministic distributed algorithm for general graphs uses a distributed version of the well-known *median-of-median* technique, which we briefly outlined in Section 2, resulting in a time complexity of  $O(Dn^{0.9114})$  for a constant group size. A straightforward modification of this algorithm in which the group size in each phase  $i$  is set to  $O(\sqrt{n^{(i)}})$  results in a much better time complexity. It can be shown that the time complexity of this variant of the algorithm is bounded by  $O(D(\log n)^{\log \log n + O(1)})$ . However, since our proposed algorithm is substantially better, we will not further discuss this median-of-median-based algorithm. Due to the more complex nature of the deterministic algorithm, we will treat the randomized algorithm first.

##### 4.1 Randomized algorithm

While the derivation of an expedient deterministic algorithm is somewhat intricate, it is remarkably simple to come up with a fast randomized algorithm. An apparent solution, proposed by Shrira et al.,<sup>20</sup> is to choose a node randomly and take its element as an initial guess. After computing the number of nodes with smaller and larger elements, it is likely that a considerable fraction of all nodes no longer need be considered. By iterating this procedure on the

remaining candidate nodes, the  $k^{\text{th}}$  smallest element can be found quickly for all  $k$ .

A node can be chosen randomly using the following scheme: A message indicating that a random element is to be selected is sent along a random path in the spanning tree starting at the root. If the root has  $l$  children  $v_1, \dots, v_l$  where child  $v_i$  is the root of a subtree with  $n_i$  candidate nodes including itself, the root chooses its own element with probability  $1/(1 + \sum_{j=1}^l n_j)$ . Otherwise, it sends a message to one of its children. The message is forwarded to node  $v_i$  with probability  $n_i/(1 + \sum_{j=1}^l n_j)$  for all  $i \in \{1, \dots, l\}$ , and the recipient of the message proceeds in the same manner. It is easy to see that this scheme selects a node uniformly at random and that it requires at most  $2D$  time, because the times to reach any node and to report back are both bounded by  $D$ . Note that after each phase the probabilities change as they depend on the altered number of candidate nodes remaining in each subtree. However, having determined the new interval in which the solution must lie, the number of nodes satisfying the new predicate in all subtrees can again be computed in  $2D$  time.

This straightforward procedure yields an algorithm that finds the  $k^{\text{th}}$  smallest element in  $O(D \log n)$  expected time, as  $O(\log n)$  phases suffice in expectation to narrow down the number of candidates to a small constant. It can even be shown that the time required is  $O(D \log n)$  with high probability. The key observation to improve this algorithm is that picking a node randomly always takes  $O(D)$  time, therefore several random elements ought to be chosen in a single phase in order to further reduce the number of candidate nodes. The method to select a single random element can easily be modified to allow for the selection of several random elements by including the number of needed random elements in the request message. A node receiving such a message locally determines whether its own element is chosen, and also how many random elements each of its children's subtrees has to provide. Subsequently, it forwards the requests to all of its children whose subtrees must produce at least one random element. Note that all random elements can be found in  $D$  time independent of the number of random elements, but due to the restriction that only a constant number of elements can be packed into a single message, it is likely that not all elements can propagate back to the root in  $D$  time. However, all elements still arrive at the root in  $O(D)$  time if the number of random elements is bounded by  $O(D)$ .

By using this simple *pipelining* technique to select  $O(D)$  random elements in  $O(D)$  time, we immediately get a more efficient algorithm, which we will henceforth refer to as  $A^{\text{rand}}$ . When selecting  $O(D)$  elements uniformly at random in each phase, it can be shown that the number of candidates is reduced by a factor of  $O(D)$  in a constant number of phases with high probability with respect to  $O(D)$ , as opposed to merely a constant factor in case only a single element is chosen. This result, together with the observation that each phase costs merely  $O(D)$  time, is used to prove the following theorem.

**THEOREM 4.1.** *In a connected graph of diameter  $D \geq 2$  consisting of  $n$  nodes, the time complexity of algorithm  $A^{\text{rand}}$  is  $O(D \log_D n)$  w.h.p.*

In particular in graphs where  $D$  is large, algorithm  $A^{rand}$  is considerably faster than the algorithm selecting only a single random element in each phase. In Section 5, we prove that no deterministic or probabilistic algorithm can be better asymptotically, i.e.,  $A^{rand}$  is *asymptotically optimal*.

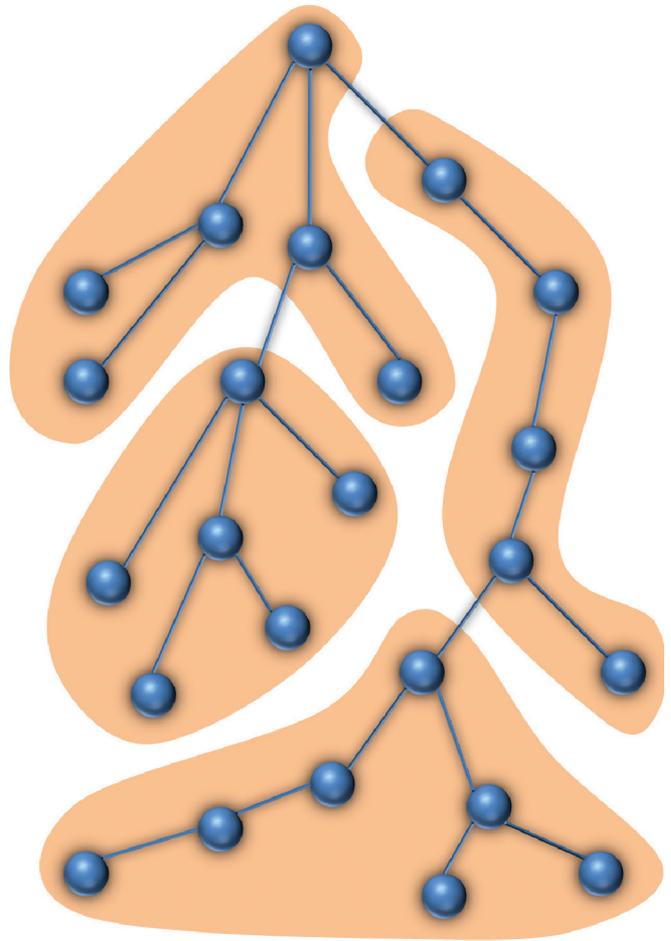
#### 4.2 Deterministic algorithm

The difficulty of deterministic iterative algorithms for  $k$ -selection lies in the selection of elements that provably allow for a reduction of the search space in each phase. Once these elements have been found, the reduced set of candidate nodes can be determined in the same way as in the randomized algorithm. Thus, the deterministic algorithm, referred to as  $A^{det}$ , has to compute a set of  $O(D)$  elements that partitions all elements similarly to the random set used by algorithm  $A^{rand}$  in each phase.

A simple idea to go about this problem is to start sending up elements from the leaves of the spanning tree, accumulating the elements from all children at the inner nodes, and then recursively forwarding a selection of  $t$  elements to the parent. The problem with this approach is the reduction of all elements received from the children to the desired  $t$  elements. If a node  $v_i$  receives  $t$  elements from each of its  $c_i$  children in the spanning tree, the  $t$  elements that partition all  $c_i t$  nodes into segments of approximately equal size ought to be found. However, in order to find these elements, the number of elements in each segment has to be counted starting at the leaves. Since this counting has to be repeated in each step along the path to the root, the time required to find a useful partitioning into  $k$  segments requires  $O(D(D + Ct))$  time, where  $C := \max_{i \in \{1, \dots, n\}} c_i$ . This approach suffers from several drawbacks: It takes at least  $O(D^2)$  time just to find a partitioning, and the time complexity depends on the structure of the spanning tree.

Our proposed algorithm  $A^{det}$  solves these issues in the following manner. In any phase  $i$ , the algorithm splits the entire spanning tree into  $O(\sqrt{D})$  groups, each of size  $O(n^{(i)}/\sqrt{D})$ . Figure 2 depicts an example tree split into 4 groups. Recursively, in each of those groups a particular node initiates the same partitioning into  $O(\sqrt{D})$  groups as long as the group size is larger than  $O(\sqrt{D})$ . The goal of this recursive partitioning is to find, for each group,  $O(\sqrt{D})$  elements that reduce the search space by a factor of  $O(\sqrt{D})$ . Groups of size at most  $O(\sqrt{D})$  can simply report all their elements to the node that initiated the grouping at this recursion level. Once such an initiating node  $v$  has received all  $O(\sqrt{D})$  elements from each of the  $O(\sqrt{D})$  groups it created, it sorts those  $O(D)$  elements, and subsequently issues a request to count the nodes in each of the  $O(D)$  intervals induced by the received elements. Assume that all the groups created by node  $v$  together contain  $n_v^{(i)}$  nodes in phase  $i$ . The intervals can locally be merged into  $O(\sqrt{D})$  intervals such that each interval contains at most  $O(n_v^{(i)}/\sqrt{D})$  nodes. These  $O(\sqrt{D})$  elements are recursively sent back to the node that created the group to which node  $v$  belongs. Upon receiving the  $O(D)$  elements from its  $O(\sqrt{D})$  groups and counting the number of nodes in each interval, the root can initiate phase  $i + 1$  for which it holds that  $n^{(i+1)} < n^{(i)}/O(\sqrt{D})$ .

**Figure 2: An example tree of diameter 12 consisting of 24 nodes is split according to algorithm  $A^{det}$  into 4 groups, each consisting of at most 7 nodes.**



Given that the number of candidates reduces by a factor of  $O(\sqrt{D})$  in each phase, it follows that the number of phases is bounded by  $O(\log_D n)$ . It can be shown that each phase costs  $O(D \log_D n)$  time, which proves the following bound on the time complexity.

**THEOREM 4.2.** *In a connected graph of diameter  $D \geq 2$  consisting of  $n$  nodes, the time complexity of algorithm  $A^{det}$  is  $O(D \log_D^2 n)$ .*

#### 5. LOWER BOUND

In this section, we sketch how to prove a time lower bound for *generic* distributed selection algorithms which shows that the time complexity of the simple randomized algorithm of Section 4.1 for finding the element of rank  $k$  is asymptotically optimal for most values of  $k$ . Informally, we call a selection algorithm generic if it does not exploit the structure of the element space except for using the fact that there is a global order on all the elements. Formally, this means that the only access to the structure of the element space is by means of the comparison function. Equivalently, we can assume that all elements assigned to the nodes are fixed but that the ordering of elements belonging to different nodes is determined by an adversary and is initially not known to the

nodes. For the lower bound, we use a simpler synchronous communication model where time is divided into rounds and in every round each node can send a message to each of its neighbors. Note that since the synchronous model is strictly more restrictive than the asynchronous model, a lower bound for the synchronous model directly carries over to the asynchronous model. We show that if in any round only one element can be transmitted over any edge, such an algorithm needs at least  $\Omega(D \log_b n)$  rounds to find the median with reasonable probability.

Generalizing the lower bound to finding the element of rank  $k$  for arbitrary  $k \in \{1, \dots, n\}$  is straight-forward. We can assume that  $k \leq n/2$  because finding the element of rank  $k$  is equivalent to finding the element of rank  $n + 1 - k$  with respect to the inverse global order. We can now just inform the algorithm about the rank of all but the first  $2k$  elements (additional information cannot make the problem harder). The problem now reduces to finding the median of  $2k$  elements.

Let us start with an outline of our proof strategy. The lower bound is proven in two steps. We first consider protocols between two nodes where each of the nodes starts with half of the elements. Assuming that the two nodes can send each other messages containing at most  $B \geq 1$  elements in each round, we show that  $\Omega(\log_B n)$  rounds are needed to find the median. In a second step, we obtain the desired lower bound for general graphs by means of a reduction: We construct a graph  $G(D)$  for every diameter  $D \geq 3$  such that every  $T$ -round median algorithm on  $G(D)$  can be turned into a  $T/(D - 2)$ -round two-party protocol in which the two nodes have to communicate  $D - 2$  elements per message.

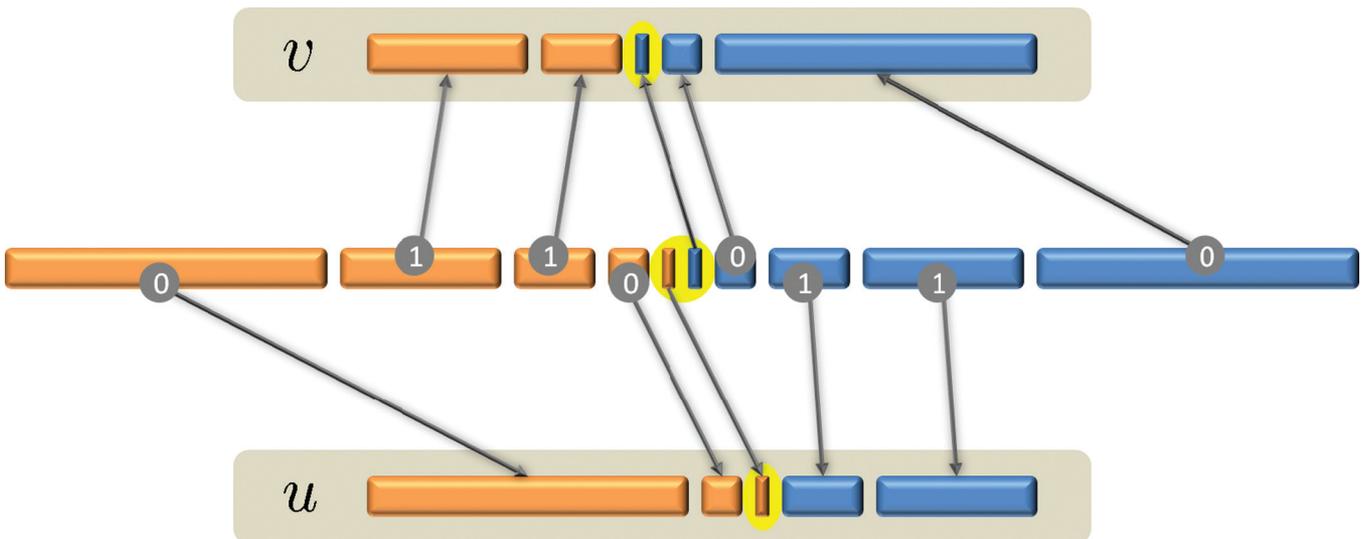
We therefore start by studying protocols between two nodes  $u$  and  $v$  such that  $u$  and  $v$  each have  $N \geq 1$  elements  $u_0 < u_1 < \dots < u_{N-1}$  and  $v_0 < v_1 < \dots < v_{N-1}$  respectively, where  $<$  is the global order according to which we want to find the median. We denote the sets of elements of  $u$  and  $v$  by  $S_u$  and  $S_v$ , respectively. Each message  $M = (S, X)$  between the two nodes is further assumed to contain a set  $S$  of at

most  $B$  elements and some arbitrary additional information  $X$ . Assume  $M$  is a message from  $u$  to  $v$ . In this case,  $X$  can be everything which can be computed from the results of the comparisons between all the elements  $u$  has seen so far, as well as all the additional information  $u$  has received so far. The only restriction on  $X$  is that it cannot be used to transmit information about the values of elements not in  $S$  or in one of the earlier messages. We call a protocol between  $u$  and  $v$  which only sends messages of the form  $M = (S, X)$  as described above, a generic two-party protocol.

The general idea is as follows. We define  $N$  different partitions, each assigning  $N$  of the  $2N$  elements to  $u$ , and the other  $N$  elements to  $v$  (i.e.,  $N$  different orders between the elements in  $S_u$  and  $S_v$ ) in such a way that each partition results in a different median element. We choose as input one of the  $N$  partitions uniformly at random. In order to compute the median, we then have to find out which of the  $N$  partitions was chosen. We show that in each communication round, the probability for reducing the number of possible partitions by more than a factor of  $\lambda B$  is exponentially small in  $\lambda$ .

For simplicity, assume that  $N = 2^l$  is a power of 2. Let  $X_0, \dots, X_{l-1} \sim \text{Bernoulli}(1/2)$  be  $l$  independent Bernoulli variables, i.e., all  $X_i$  take values 0 or 1 with equal probability. The partition of the  $2N$  elements among  $u$  and  $v$  is determined by the values of  $X_0, \dots, X_{l-1}$ . If  $X_{l-1} = 0$ , the  $N/2$  smallest of the  $2N$  elements are assigned to  $u$  and the  $N/2$  largest elements are assigned to  $v$ . If  $X_{l-1} = 1$ , it is the other way round. In the same way, the value of  $X_{l-2}$  determines the assignment of the smallest and largest  $N/4$  of the remaining elements: If  $X_{l-2} = 0$ ,  $u$  gets the elements with ranks  $N/2 + 1, \dots, 3N/4$  and  $v$  gets the elements with ranks  $5N/4 + 1, \dots, 3N/2$  among all  $2N$  elements. Again, the remaining elements are recursively assigned analogously depending on the values of  $X_{l-3}, \dots, X_0$  until only the two elements with ranks  $N$  and  $N + 1$  (i.e., the two median elements) remain. The element with rank  $N$  is assigned to  $u$  and the element with rank  $N + 1$  is assigned to  $v$ . Figure 3 illustrates the described process.

**Figure 3:** The  $2N$  elements minus the two medians are assigned to  $u$  and  $v$  according to  $l = \log n$  independent Bernoulli variables  $X_0, \dots, X_{l-1}$ . One of the two medians is assigned to  $u$  and the other to  $v$ . In order to find the medians,  $u$  and  $v$  must compute the values of  $X_0, \dots, X_{l-1}$ .



Consider the two elements  $u_{\alpha^*}$  and  $v_{\beta^*}$  with ranks  $N$  and  $N + 1$ , respectively, and let  $\alpha^*$  and  $\beta^*$  be the ranks of the elements  $N$  and  $N + 1$  within the sets  $S_u$  and  $S_v$ . We consider the median problem to be solved as soon as either  $u$  knows  $\alpha^*$  or  $v$  knows  $\beta^*$ . The elements are partitioned in such a way that the random variables  $X_i$  directly determine the base-2 representations of  $\alpha^*$  and  $\beta^*$ . If  $X_0 = 0$ , the most significant bit of the bit representation of  $\alpha^*$  is 1, whereas the most significant bit of the bit representation of  $\beta^*$  is 0. If  $X_0 = 1$ , the most significant bit of  $\alpha^*$  is 0 and the most significant bit of  $\beta^*$  is 1. The other bits of the base-2 representations of  $\alpha^*$  and  $\beta^*$  are determined analogously: If  $X_i = 0$ , the  $(i + 1)$ st-most significant bit of  $\alpha^*$  is 1 and the  $(i + 1)$ st-most significant bit of  $\beta^*$  is 0 and vice versa if  $X_i = 1$ . Consider two arbitrary elements  $u_\alpha \in S_u$  and  $v_\beta \in S_v$  with ranks  $\alpha$  and  $\beta$  within the two sets  $S_u$  and  $S_v$ , respectively. The outcome of the comparison of  $u_\alpha$  and  $v_\beta$  (i.e., whether  $u_\alpha < v_\beta$  or  $v_\beta < u_\alpha$ ) is determined by the first variable  $X_i$  that is equal to the corresponding bit in the base-2 representation of  $u_\alpha$  or different from the corresponding bit in the base-2 representation of  $v_\beta$ , whatever occurs first. If  $X_i = 0$ , we have that  $u_\alpha < v_\beta$ , otherwise  $v_\beta < u_\alpha$ .

Clearly,  $u$  and  $v$  can only learn about  $\alpha^*$  and  $\beta^*$  from comparisons between their own elements and elements they have received from the other node and from additional information that the nodes sent to each other. Consider a generic two-party algorithm  $A$  that computes the median. Assume that after a certain time of the execution of  $A$ ,  $u_{\alpha_1}, \dots, u_{\alpha_r}$  are the elements that  $u$  has sent to  $v$  and  $u_{\beta_1}, \dots, u_{\beta_s}$  are the elements that  $v$  has sent to  $u$ . Let  $\hat{i}$  be the largest index such that there is an element  $u_{\beta_{\hat{i}}}$  for which the first  $\hat{i}$  bits are different from the corresponding variable  $X_i$  or such that there is an element  $v_{\beta_{\hat{i}}}$  for which the first  $\hat{i}$  bits are equal to the corresponding variable  $X_i$ . By the above observation, any comparison between an element in  $S_u$  and an element that  $v$  has sent to  $u$  and any comparison between an element in  $S_v$  and an element that  $u$  has sent to  $v$  is determined by the values of  $X_0, \dots, X_{\hat{i}-1}$ . Intuitively,  $u$  and  $v$  cannot have any information about the values of  $X_{\hat{i}+1}, \dots, X_{l-1}$ . Thus,  $u$  and  $v$  have to guess the remaining bits by sending each other the right elements. It can be shown that the probability for guessing at least  $\xi$  bits correctly in a single round is at most  $2B/2^\xi$ . The number of newly learned bits in each round can be upper bounded by independent random variables. Using a Chernoff-type argument, one can then show that  $\log_{2B}(N)/c$  rounds are needed to learn all  $l$  bits  $X_0, \dots, X_{l-1}$  with probability at least  $1 - 1/N^{\frac{1+c}{c}}$ , implying the following theorem.

**THEOREM 5.1.** *Every, possibly randomized, generic two-party protocol to find the median needs at least  $\Omega(\log_{2B} N)$  rounds in expectation and with probability at least  $1 - 1/N^\delta$  for every constant  $\delta < 1/2$ .*

Based on the lower bound for two-party protocols, we can now prove a lower bound for generic selection algorithms on general graphs. In the following, we assume that every node of a graph with  $n$  nodes starts with one element and that we have to find the  $k^{\text{th}}$  smallest of all  $n$  elements. In every round, every node can send one element to each of its neighbors.

For every  $n \geq D \geq 3$ , we construct a graph  $G(D)$  with  $n$  nodes and diameter  $D$  such that we can reduce the problem of finding the median by a two-party protocol to the problem of finding the element of rank  $k$  in  $G(D)$ .

We first describe a lower bound of  $\Omega(D \log_D n)$  for finding the median and then generalize to finding the element of an arbitrary rank  $k$ . For simplicity, assume that  $n - D$  is an odd number. Let  $N = (n - D + 1)/2$ . We consider the graph  $G(D)$  defined as follows: The graph  $G(D)$  consists of two nodes  $u$  and  $v$  that are connected by a path of length  $D - 2$  (i.e., it contains  $D - 1$  nodes). In addition, there are nodes  $u_1, \dots, u_N$  and  $v_1, \dots, v_N$  such that  $u_i$  is connected to  $u$  and  $v_i$  is connected to  $v$  for all  $i \in \{1, \dots, N\}$ . We can certainly assume that  $n = \omega(D)$  because  $\Omega(D)$  is a trivial lower bound (even finding the minimum element requires  $\Omega(D)$  rounds). We can therefore assume that only the leaf nodes  $u_i$  and  $v_i$  for  $i \in \{1, \dots, N\}$  hold an element and that we need to find the median of these  $2N$  elements. We can simply assign dummy elements to all other nodes such that the global median is equal to the median of the leaf elements. Since only the leaves start with an element, we can assume that in the first round, all leaves  $u_i$  send their element to  $u$  and all leaves  $v_i$  send their element to  $v$ , as this is the only possible useful communication in the first round. By this, the problem reduces to finding the  $k^{\text{th}}$  smallest element of  $2N$  elements on a path of length  $D - 2$  if initially each of the two end nodes  $u$  and  $v$  of the path holds  $N$  elements. Note that the leaf nodes of  $G(D)$  do not need to further participate in a distributed selection protocol since  $u$  and  $v$  know everything their respective leaves know and can locally simulate all actions of their leaf nodes.

Assume that we are given an algorithm  $A$  that finds the median on  $G(D)$  in time  $T + 1$ . We sketch how to construct a two-party protocol  $A'$  that sends at most  $D - 2$  elements per message and finds the median in time  $\lceil T/(D - 2) \rceil$ . The  $\Omega(\log_D N)$  lower bound for such an algorithm  $A'$  then implies that  $T = \Omega(D \log_D N)$ . Because information needs at least  $D - 2$  to travel from  $u$  to  $v$  and vice versa, everything  $u$  and  $v$  can compute in round  $t$  (i.e., after receiving the message from round  $t - 1$ ) is a function of their own elements and the content of the messages of the other node up to round  $t - (D - 2)$ . For example,  $u$ 's messages of the first  $D - 2$  rounds only depend on  $S_u$ , the messages of rounds  $D - 1, \dots, 2(D - 2)$  can be computed from  $S_u$  and  $v$ 's messages in the first  $D - 2$  rounds, and so on. To obtain a two-party protocol  $A'$  from  $A$ , we can proceed as follows. In the first round of  $A'$ ,  $u$  and  $v$  send their messages of the first  $D - 2$  rounds of  $A$  to each other. Now,  $u$  and  $v$  can both locally simulate all communication of the first  $D - 2$  rounds of  $A$ . This allows to compute the messages of the next  $D - 2$  rounds of  $A$ . In general, in round  $r$  of the two-party protocol  $A'$ ,  $u$  and  $v$  can send each other their messages of rounds  $(r - 1)(D - 2) + 1, \dots, r(D - 2)$  of  $A$  and can afterwards locally simulate the respective rounds of  $A$ . The time complexity of  $A'$  then becomes  $\lceil T/(D - 2) \rceil$ .

**THEOREM 5.2.** *For every  $n \geq D \geq 3$ , there is a graph  $G(D)$  with  $n$  nodes and diameter  $D$  such that every, possibly randomized, generic algorithm to find the  $k^{\text{th}}$  smallest element requires*

$\Omega(D \log_D \min\{k, n - k\})$  rounds in expectation and with probability at least  $1 - 1/(\min\{k, n - k\})^\delta$  for every constant  $\delta < 1/2$ . In particular, finding the median requires at least  $\Omega(D \log_D n)$  rounds.

Similar proof techniques have been used in the area of communication complexity where people try to find bounds on the total number of bits that have to be transmitted to solve a certain communication problem.<sup>22</sup> In particular,<sup>21</sup> introduces a simulation technique to run two-party protocols in paths and more general graphs in order to extend lower bound for computations between two nodes to computations on more general topologies. In contrast to communication complexity, our focus is on minimizing the number of communication rounds rather than minimizing the number of transmitted bits.

## 6. CONCLUSION

In this article, we studied the  $k$ -selection problem, a prominent data aggregation problem, and proved upper and lower bounds on its (time) complexity. Our results are presented in an entirely abstract way, i.e., it remains to show that our algorithms have a notable impact on the performance of aggregation functions in real networks and thus prove to be relevant in practical applications. Apparently, it is usually not possible to simply implant a distributed algorithm in an application domain, as additional constraints imposed by the application need to be respected. In wireless sensor networks, e.g., aggregation needs to adhere to wireless channel characteristics. We believe that our work can shed additional light on the achievable aggregation rate and capacity<sup>10,11</sup> in wireless networks, or, combined with other algorithmic work on data gathering such as, e.g.,<sup>5,6</sup> may even provide basic aggregation functionality for various application domains. We hope that our results and techniques may eventually find their way into several application areas, providing aggregation support for, e.g., streaming databases or multi-core architectures.

## Acknowledgments

We would like to thank Pascal von Rickenbach and Roland Flury for their help with the illustrations. Moreover, we would like to express our gratitude to Hagit Attiya for providing valuable feedback, which helped us greatly to improve this article, and also for finding the time to write a technical perspective.

The original version of this paper is entitled “Tight Bounds for Distributed Selection” and can be found in the *Proceedings of the 19th ACM Symposium on Parallelism in Algorithms and Architectures* (San Diego, CA, June 2007). 

## References

- Blum, M., Floyd, R. W., Pratt, V., Rivest, R. L., and Tarjan, R. E. Time bounds for selection. *Journal of Computer and System Sciences*, 7:448–461, 1973.
- Burri, N., von Rickenbach, P., and Wattenhofer, R. Dozer. Ultra-low power data gathering in sensor networks. In *International Conference on Information Processing in Sensor Networks (IPSN)*, 2007.
- Chin, F. Y. L. and Ting, H. F. An improved algorithm for finding the median distributively. *Algorithmica*, 2:77–86, 1987.

- Frederickson, G. N. Tradeoffs for selection in distributed networks. In *Proceedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 154–160, 1983.
- Goel, A. and Estrin, D. Simultaneous optimization for concave costs: Single sink aggregation or single source buy-at-bulk. *Algorithmica*, 43(1–2):5–15, 2005.
- Jia, L., Lin, G., Noubir, G., Rajaraman, R., and Sundaram, R. Universal approximations for TSP, Steiner Tree, and set cover. In *37th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 386–395, 2005.
- Kempe, D., Dobra, A., and Gehrke, J. Gossip-based computation of aggregate information. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2003.
- Madden, S., Franklin, M. J., Hellerstein, J. M., and Hong, W. TAG: a tiny aggregation service for ad-hoc sensor networks. In *Proceedings of the 5th Annual Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 131–146, 2002.
- Marberg, J. M. and Gafni, E. An optimal shout-echo algorithm for selection in distributed sets. In *Proceedings of the 23rd Allerton Conference on Communication, Control, and Computing*, 1985.
- Moscibroda, T. The worst-case capacity of wireless sensor networks. In *6th International Conference on Information Processing in Sensor Networks (IPSN)*, 2007.
- Moscibroda, T. and Wattenhofer, R. The complexity of connectivity in wireless networks. In *25th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2006.
- Negro, A., Santoro, N., and Urrutia, J. Efficient distributed selection with bounded messages. *IEEE Transactions on Parallel and Distributed Systems*, 8(4):397–401, 1997.
- Patt-Shamir, B. A note on efficient aggregate queries in sensor networks. In *Proceedings of the 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 283–289, 2004.
- Peleg, D. *Distributed Computing: A Locality-Sensitive Approach*. SIAM Monographs on Discrete Mathematics and Applications, 2000.
- Rodeh, M. Finding the median distributively. *Journal of Computer and System Science*, 24(2):162–166, 1982.
- Rodem, D., Santoro, N., and Sidney, J. Shout-echo selection in distributed files. *Networks*, 16:235–249, 1986.
- Santoro, N., Scheutzw, M., and Sidney, J. B. On the expected complexity of distributed selection. *Journal on Parallel and Distributed Computing*, 5(2):194–203, 1988.
- Santoro, N., Sidney, J. B., and Sidney, S. J. A distributed selection algorithm and its expected communication complexity. *Theoretical Computer Science*, 100(1):185–204, 1992.
- Schönhage, A., Paterson, M. S., and Pippenger, N. Finding the median. *Journal of Computer and System Sciences*, 13:184–199, 1976.
- Shrira, L., Francez, N., and Rodeh, M. Distributed  $k$ -selection: From a sequential to a distributed algorithm. In *Proceedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 143–153, 1983.
- Tiwari, P. Lower bounds on communication complexity in distributed computer networks. *Journal of the ACM (JACM)*, 34(4):921, 1987.
- Yao, A. Some complexity questions related to distributive computing. In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC)*, pp. 209, 1979.
- Yao, Y. and Gehrke, J. The Cougar approach to in-network query processing in sensor networks. *ACM SIGMOD Record*, 31(3):9–18, 2002.
- Zhao, J., Govindan, R., and Estrin, D. Computing aggregates for monitoring wireless sensor networks. In *Proceedings of the 1st IEEE International Workshop on Sensor Network Protocols and Applications (SNPA)*, 2003.

**Fabian Kuhn** (kuhn@inf.ethz.ch) Postdoc researcher, Institute of Theoretical Computer Science, ETH Zurich, Switzerland

**Thomas Locher** (lochert@tik.ee.ethz.ch) PhD student, Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland

**Roger Wattenhofer** (wattenhofer@tik.ee.ethz.ch) Professor, Head of Distributed Computing Group, Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland

© 2008 ACM 0001-0782/08/0900 \$5.00