

On the Complexity of the “Most General” Firing Squad Synchronization Problem

Darin Goldstein* and Kojiro Kobayashi†

Abstract

We show that if a minimal-time solution exists for a fundamental distributed computation primitive, synchronizing a general directed network of finite-state processors, then there must exist an extraordinarily fast $O(ED \log_2 D (\log_2 n)^2)$ algorithm in the RAM model of computation for exactly determining the diameter of a general directed graph. The proof is constructive.

This result interconnects two very distinct areas of computer science: distributed protocols on networks of intercommunicating finite-state machines and standard algorithms on the usual RAM model of computation.

1 Introduction

The Firing Squad Synchronization Problem (or FSSP, for short) is a famous problem originally posed almost half a century ago. A prisoner is about to be executed by firing squad. The firing squad is made up of soldiers who have formed up in a straight line with muskets aimed at the prisoner. The general stands on the left side of the line, ready to give the order, but he knows that he can only communicate with the soldier to his right. In fact, each soldier can only communicate with the soldier to his immediate left and/or right, but nobody else. Soldiers have limited memory and can only pass along simple instructions. Is it possible to come up with a protocol, independent of the size of the line, for getting all of the soldiers to fire at the prisoner simultaneously if their only means of communication are small, whispered instructions only to adjacent soldiers? (The possibility of counting the number of soldiers in the line can be discounted because no soldier can remember such a potentially large amount of information; each soldier can only remember messages that are independent of the size of the line.)

The problem itself is interesting as a mathematical puzzle. More importantly, there are also applications to the synchronization of small, fast processors in large networks. In the literature on the subject (e.g. [26, 18]), the problem has been referred to as “macrosynchronization given microsynchronization” and “realizing global synchronization using only local information exchange.” The synchronization of multiple small but fast processors in general networks is a fundamental problem of parallel processing and a computing primitive of distributed computation.

The FSSP has a rich history that spans decades. J. Mazoyer provides an overview of the problem (up to 1986, at least) in addition to some of its history in [20]. The problem was originally introduced by J. Myhill in 1957, though the first published reference is [23] from 1962. J. McCarthy and M. Minsky [22] first solved the problem. Since then, many variations of FSSP have been proposed and studied (e.g. [21, 28, 19, 24, 12, 3, 2] to name just a few). Despite this long history of study and extensive compilation of results, one fundamental problem for FSSP has remained open. It concerns existence of optimal-time solutions of variations of FSSP.

*Department of Computer Engineering and Computer Science, California State University, Long Beach, *dar-ing@cecs.csulb.edu*

†Department of Information Systems Science, Faculty of Engineering, Soka University, *kobayashi@t.soka.ac.jp*

For one variation of FSSP that has a solution and a problem instance (a network) X of the variation, the *minimum firing time* of X is the minimum of the firing times of solutions A for the instance X , where A ranges over all solutions of the variation. A *minimal-time solution* of the variation is a solution whose firing time for X is the minimum firing time of X for any problem instance X of the variation. Variations of FSSP are classified to the following three types:

Type A: The variation has a minimal-time solution.

Type B: The variation has a solution but has no minimal-time solution.

Type C: The variation has no solution.

We know many examples of variations of Type A. The original FSSP is one of them. The minimum firing time of a linear array of n soldiers is $2n - 2$ and minimal-time solutions were found by Goto [11] and Waksman [29]. Other examples are linear arrays with arbitrary position of the general, bilateral rings, one-way rings, rectangles, and squares (the position of the general may be either one of the corners or may be at any position). We also know examples of variations of Type C. One is the variation for directed networks such that an out-port may have arbitrary fan-out ([17]) and another is the variation for undirected networks with unbounded number of generals ([14, 13]). However, at present we know no example of variations of Type B. Neither do we know that variations of Type B do not exist. Therefore, the problem to decide whether variations of Type B exist or not is a very important open problem. Here we restrict ourselves to variations of FSSP that are obtained from the original FSSP by generalization only with respect to network topologies. Without this restriction, we know of some variations of Type B: [21] (generalization with respect to the communication delay time) and [27] (generalization with respect to the number and the role of generals).

In the followings, we list eight variations for which solutions are known but minimal-time solutions are not known.

2PATH: The variations for paths in the two-dimensional grid space. A path may bend but may not touch itself. The general is at one of the end points.

g-2PATH: The same as 2PATH but the general may be at any position (g is for “generalized”).

2REG: The variation for finite connected regions in the two-dimensional grid space. The general may be at any position. Regions may have holes.

3PATH, g-3PATH, 3REG: The variations analogous to 2PATH, g-2PATH, 2REG respectively for the three-dimensional grid space.

UN: The variation for general undirected networks. Networks must be connected.

DN: The variation for general directed networks. Networks must be strongly connected.

We know a solution with firing time $3r - 1$ for UN ([25]) (r denotes the radius of the network) and solutions for DN with an exponential firing time ([16]), with a firing time $O(n^2)$ ([7]), and with a firing time $O(Dn)$ ([26], D denotes the diameter of the network). However, these solutions are not minimal-time. The variation DN is especially important because it is the “most general” FSSP.

We have made the conjecture that all of these eight variations have no minimal-time solutions, and hence are of Type B. The attempt to prove this conjecture was initiated by one of the authors. In [18], the second author introduced a problem which he called 2PEP (the two-dimensional path extension problem). The problem is to decide, for each given path in the two-dimensional grid space, whether we can extend the path from the specified end point so that the length of the path is doubled. At present we know no polynomial time algorithm for 2PEP, but at the same time we are unable to prove that it is NP-complete. It was proved

in [18] that if 2PATH has a minimal-time solution then 2PEP has an $O(n^2)$ time algorithm. This result readily applies also to g-2PATH and 2REG. Hence, finding a minimal-time solution of 2PATH, g-2PATH, 2REG is at least as hard as finding an $O(n^2)$ time algorithm for 2PEP. Next, in [10] and [9], we could prove a stronger result for the other three variations: if $P \neq NP$ then 3PATH, g-3PATH, 3REG have no minimal-time solutions. Hence, finding a minimal-time solution of 3PATH, g-3PATH, 3REG is at least as hard as finding a polynomial time algorithm for SAT. UN and DN remain to be studied.

In the present paper, we show that finding a minimal-time solution of DN is at least as hard as finding an “incredibly” fast algorithm in the RAM model of computation for the problem to determine the diameter of a general directed graph. We formalize this with the following theorem.

Theorem 1 *If there exists a minimal-time solution for the general directed network topology, then there exists a deterministic algorithm in the RAM model of computation that can exactly determine the diameter of a general unweighted directed graph in time $O(ED \log_2 D (\log_2 n)^2)$ where E is the number of edges, D is the diameter of the graph, and $n = |V|$ is the number of vertices.*

| | $O(En + n^2 \log n)$ | $O(ED \log D (\log n)^2)$ |
|---|----------------------|------------------------------|
| $E = \Theta(n^2), D = \Theta(n)$ | $O(n^3)$ | $O(n^3 (\log n)^3)$ |
| $E = \Theta(n^2), D = \Theta(\sqrt{n})$ | $O(n^3)$ | $O(n^2 \sqrt{n} (\log n)^3)$ |
| $E = \Theta(n^2), D = \Theta(1)$ | $O(n^3)$ | $O(n^2 (\log n)^2)$ |
| $E = \Theta(n\sqrt{n}), D = \Theta(n)$ | $O(n^2 \sqrt{n})$ | $O(n^2 \sqrt{n} (\log n)^3)$ |
| $E = \Theta(n\sqrt{n}), D = \Theta(\sqrt{n})$ | $O(n^2 \sqrt{n})$ | $O(n^2 (\log n)^3)$ |
| $E = \Theta(n\sqrt{n}), D = \Theta(1)$ | $O(n^2 \sqrt{n})$ | $O(n\sqrt{n} (\log n)^2)$ |
| $E = \Theta(n), D = \Theta(n)$ | $O(n^2 \log n)$ | $O(n^2 (\log n)^3)$ |
| $E = \Theta(n), D = \Theta(\sqrt{n})$ | $O(n^2 \log n)$ | $O(n\sqrt{n} (\log n)^3)$ |
| $E = \Theta(n), D = \Theta(1)$ | $O(n^2 \log n)$ | $O(n (\log n)^2)$ |

Table 1: Comparison of the running times of Dijkstra’s algorithm and the algorithm of Theorem 1

To give some idea on how the algorithm mentioned in Theorem 1 compares with other algorithms, in Table 1 we show the running time $O(En + n^2 \log n)$ of Dijkstra’s algorithm and the running time $O(ED \log D (\log n)^2)$ of our algorithm for some special cases of E, D . Our algorithm is slower only for the case $D = \Theta(n)$ (narrow and long graphs).

The literature for algorithms that determine the diameter of a graph is vast and varied. There are *numerous* references for finding the diameter of directed graphs, most of which reduce to the problems of matrix multiplication (that ignores addition as an elementary operation when determining the complexity, which our model does not) or solving the All Pairs Shortest Path (APSP) problem; the problem of finding approximate diameters (e.g. [1]) has also been examined. To date, the best methods for determining the diameter of an arbitrary directed network that do not involve “fast matrix multiplication” [4] rely on solving the All Pairs Shortest Path problem [5]. Applying Dijkstra’s algorithm for each vertex in the network leads to a running time of $O(En + n^2 \log n)$. This algorithm was improved by Karger et al. [15] in 1993 to $O(nE^* + n^2 \log n)$ where E^* represents the total number of edges that participate in the shortest length paths. Fredman [8] in 1976 introduced a method in the algebraic computation tree model by which the min-plus matrix multiplication necessary to compute the APSP could be determined using only $O(n^{2.5})$ additions and comparisons, but the best known implementation by Zwick [31] in the RAM model runs in time slightly $o(n^3)$. We have presented only a very superficial sampling of the results in this area¹, but Zwick [30] presents a fairly complete survey on the APSP problem and most of its reasonable variations.

¹Indeed, there were at least six results leading up to Zwick’s recent $o(n^3)$ algorithm just for implementing Fredman’s original 1976 idea.

To the authors' knowledge, this is the first result that directly relates solutions for distributed networks of finite-state automata to algorithms that run in the "usual" RAM model of computation. To this point, there is no established connection between the diameter of a graph and the minimum firing time of a network of finite-state machines. The time bound of $O(ED \log_2 D (\log_2 n)^2)$ mentioned above in Theorem 1 beats every known result in the relevant sections of Zwick's survey by large factors, especially for somewhat sparse graphs with small diameter. One can therefore look at our results in one of two ways. One can view Theorem 1 as evidence that a minimal-time solution for the FSSP on general directed networks does not exist, given the size of the asymptotic gap between the conclusions of the theorem and the current state of research of the diameter problem after decades of searching for better algorithms. One can also view Theorem 1 as motivation for searching for a minimal-time solution, for if such a solution were found, it would be a major leap forward in the search for better "shortest-path" algorithms.

The rest of the paper will be organized as follows. In Section 2, we introduce the automata model we will use for the remainder of the paper. Section 3 contains the proof of Theorem 1 including proof sketches of the relevant lemmas. Finally, in Section 4, we present conclusions and open problems.

2 The Model

As mentioned previously, we wish to model the operation of a large network of processors whose computations are all governed synchronously by the same global clock. The model is intended to mathematically abstract a physical switching network or a very large-scale parallel processing machine. The processors are designed to be small, fast, and unable to access large memory caches. Each processor is identical and assumed to have a fixed constant number of ports which can both send and receive a constant amount of data per clock cycle. ("Constant" quantities must be independent of the size and structure of the network.)

More formally, the problem is to construct a deterministic finite-state automaton with a transition function that satisfies certain conditions. We assume that each processor in the network is identical and initially in a special "quiescent" state, in which, at each time-step, the processor sends a "blank" character through all of its ports. A processor remains in the quiescent state until a non-blank character is received by one of its in-ports. We consider connected networks of such identical synchronous finite-state automata with vertex degree uniformly bounded by a constant. These automata are meant to model very small, fast processors. The network itself may have a specific topology (see below) but potentially unbounded size. The network is formed by connecting ports of a given processor to those of other processors with wires. Not all ports of a given processor need necessarily be connected to other processors. The network has a global clock, the pulses between which each processor performs its computations. Processors synchronously, within a single global clock pulse, perform the following actions in order: read in the inputs from each of their ports, process their individual state changes, and prepare and broadcast their outputs. As mentioned, our network structure is specifically designed to model the practical situation of many small and fast processors performing a synchronous distributed computation. The goal of the protocol is to cause every process in the network to enter the same special "firing" state for the first time *simultaneously*.

A *solution* A for a given network topology (e.g. the bidirectional line, as above) is defined to be the instantiation of an automaton with a transition function that satisfies the firing conditions outlined above for any network size. (So, by this definition, a *solution* for the bidirectional line must function for a bidirectional line of **any** size.) Assuming a solution A is specified, the *firing time* of A on a given network will refer to the number of clock cycles it takes for this network of processors programmed with the solution A to complete the protocol and simultaneously fire. The *minimum firing time* of a given network (of a specified topology) will refer to the minimum over all solutions A of the firing time of the network of processors programmed with solution A . A *minimal-time solution* A_{min} for a given network topology will be a solution such that the firing time for a network of any given size (with the given topology) programmed with the algorithm A_{min} will equal the minimum firing time of the network. Note that even though the network can be of

arbitrary size, the size of the algorithm A_{min} must be fixed. The topology we consider in this paper is the most general possible, an arbitrary directed network of bounded degree².

3 The Results

To prove Theorem 1, we will first prove two lemmas.

Definition 1 *Given a graph G , define $I(v)$ to be the set of edges with terminal vertex v . For $e \in I(v')$, define $d_e(v, v')$ to be the length of the shortest path from v to v' such that the final edge in the path is e . Then for two vertices $v, v' \in G$, define $d^*(v, v')$ to be $\max_{e' \in I(v')} d_{e'}(v, v')$.*

Lemma 1 *Let $G = (V, E)$ be a general directed network with degree at most δ . Then the minimum firing time for the general directed network topology with degree bound δ running on G is $\max_{v \in V} \{d^*(root, v) + \max_{v' \in V} d(v, v')\}$.*

Proof. We need to prove two things. First, we show that any solution must take at least this long on any given network. To get a contradiction, fix a solution A that runs in time strictly less than this on some network N , and let v and v' be two vertices that achieve the above maximum. So the running time of A on N is $d^*(root, v) + d(v, v') - 1$. Let $e = (v_{prev}, v) \in I(v)$ be the edge that satisfies the maximum in the definition of d^* . Replace the edge e with a string of edges and vertices as follows: $v_{prev} \rightarrow v$ should transform into $v_{prev} \rightarrow v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_n \rightarrow v$ for some large number n . We claim that the computational transcripts of all nodes common to both networks are exactly the same through time $d^*(root, v) - 1$. Note that up until this time in both networks, the edge e has been transmitting only the quiescent value to v . Thus up until time $d^*(root, v) - 1$, all common nodes in both networks will have exactly the same computational transcripts. So because there are no other modifications, for each i , at time $d^*(root, v) + i$, the nodes at a distance i from v in network N' might have different states from their counterparts in network N . However, this implies that v' must fire at the same time in both networks. If we choose n large enough, because the argument is independent of the value of n , this is clearly impossible.

Next, we must show that, given a fixed network N , there exists a solution that fires in time at most $\max_{v, v' \in V} \{d^*(root, v) + d(v, v')\}$. Consider a solution A for FSSP that runs on general arbitrary strongly-connected digraphs. We modify this solution to get another solution A' that runs in minimal-time by running a second protocol in parallel to the original A . Release tokens that check (in a breath-first manner) whether the network follows the structure of N . Note that if we allow the tokens to keep track of the network structure that they have “seen” thus far, it is possible for each non-quiescent vertex to have pre-knowledge of when the next token should come in and what it should have thus far seen. If a token reaches or does not reach (in the case that a token is expected to arrive and does not) a vertex that is not consistent with the structure of N , then the vertex immediately sends out “USE A” breadth-first tokens. We claim that such tokens, if released from a vertex, have the time to reach every vertex in N before or at time $\max_{v \in V} \{d^*(root, v) + \max_{v' \in V} d(v, v')\}$. Assume that there exists a vertex or edge that is inconsistent with the structure of N . It will take time at most $d^*(root, v)$ for the breadth first tokens to first be released by the offending vertex v . Once released it takes time $\max_{v' \in V} \{d(v, v')\}$ for the tokens to reach every vertex in the network. Thus, either all network nodes receive “USE A” tokens or they do not before or at time $\max_{v \in V} \{d^*(root, v) + \max_{v' \in V} d(v, v')\}$. If the vertices receive a “USE A” token, they use the protocol from the original algorithm A and if not, they simply fire at time $\max_{v \in V} \{d^*(root, v) + \max_{v' \in V} d(v, v')\}$. \square

Lemma 2 *The two statements (a) “There exists a constant δ such that there exists a minimal-time solution for general directed networks with degree bounded by δ .” and (b) “For any constant δ , there exists a minimal-time solution for general directed networks with degree bounded by δ .” are equivalent.*

²The topology must be of bounded degree so that the automata can distinguish between their various in and outputs.

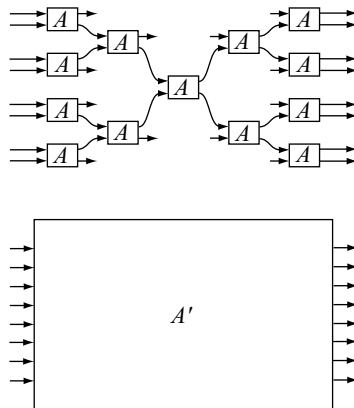


Figure 1: This figure illustrates the transformation from multiple copies of a known automata solution A to the conglomeration A' .

Proof Sketch. (b) implies (a) is obvious. We therefore concentrate on showing (a) implies (b). Fix the constant δ in statement (a). We claim that for networks with degree smaller than δ , the same solution will work because the minimum firing time of G is independent of δ by Lemma 1. For networks with degree δ' strictly greater than δ , we can form a new solution from the old solution as follows: For every node N , split N into as many nodes of degree δ as necessary. In other words, form an inward binary search tree towards the node and an outgoing binary search tree away from the node with both binary search trees degree-bounded by δ . We can now connect the wires corresponding to the links in the original network. See Figure 1. In the figure, we assume that we know the transition function of the A automata. The transition function of the automaton A' , the conglomeration of all of the A 's, can be determined as follows.

The state of A' is defined uniquely by the states of the A 's and the inputs and outputs of each A within the conglomeration. Given a vector of input values to the automata A' , the output vector after a single time step will consist of the computational transcripts of every A automata within A' if we allowed $2 \log_\delta \delta'$ “internal” time steps to elapse. Note that we are allowed to perform this simulation because both δ and δ' are constants of the network. The A' automata fires if and only if its internal A automata fire during some “internal” time step.

It is then possible to show that the firing time arising from this construction is in fact minimal via Lemma 1. The basic idea behind this calculation is to note that each edge is elongated by a factor of $2 \log_\delta \delta'$ ($\log_\delta \delta'$ inwards and $\log_\delta \delta'$ outwards) which causes the minimum firing time to gain a similar factor. However, the internal simulation of $2 \log_\delta \delta'$ time steps down to a single time step by A' compresses the running time by exactly the same factor. The actual calculations are somewhat tedious and are therefore omitted. \square

The remainder of this section will be devoted to proving Theorem 1.

If there exists a minimal-time solution for the general directed network topology, then there exists a deterministic algorithm in the RAM model of computation that can exactly determine the diameter of a general unweighted directed graph in time $O(ED \log_2 D (\log_2 n)^2)$ where E is the number of edges, D is the diameter of the graph, and $n = |V|$ is the number of vertices.

Proof (rest of Section 3). We will assume that we are given an arbitrary directed graph $G = (V, E)$ and are operating in the RAM model of computation. Tarjan’s strongly connected components algorithm can be used to determine in time $O(V + E)$ time whether the graph has infinite diameter or not. We will assume that the diameter is found to be finite. (In other words, Tarjan’s algorithm returns a single strongly connected

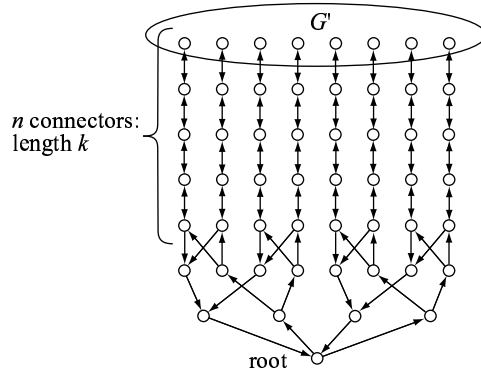


Figure 2: This figure illustrates the transformation from G' to $G''(k)$.

component.) We now proceed with a series of transformations of the graph G . Note that via Lemma 2, as long as we can show the resulting transformation is degree-bounded by some constant, it doesn't matter what the constant is.

3.1 Transforming G to G'

The first transformation takes the inputs of v , for each $v \in V$, and extends them to be of length $\frac{3}{2} \log_2 n$ that cascade inwards so as to form an incoming binary tree towards v . Note that $\log_2 n$ is sufficient for this operation; the reason for extending the natural length will become clear below. We perform a similar operation on the outgoing edges. We then connect the wires corresponding to the edges in G . The result is similar to the illustration in Figure 1 except for the fact that the final outgoing and incoming edges are extended by an additional $\frac{1}{2} \log_2 n$. Note that we have reduced the degree of the graph to at most 2. Call this transformed graph $G' = (V', E')$. We will later refer to the vertices of G inside of G' . This will refer to the set of vertices at the roots of the incoming (and hence outgoing as well) binary trees.

3.2 Transforming G' to $G''(k)$

We now create a graph $G''(k)$ from G' . First, create n bidirectional paths of length k , for some number $k \geq 3 \log_2 n$ to be determined later. For ease of communication, we will henceforward refer to these bidirectional paths as *connectors*. Connect one end of each of these connectors to a vertex in G . (Note that each vertex in G is both the end of a connector and a vertex in G' as well.) Create a root node and have the root cascade outputs outwards, each branch of length $\log_2 n$. Have these link up with the other end of the connectors. In a similar way, have edges cascade inwards towards the root, each of length $\log_2 n$, from each path linking up with the same vertices. Note that the two binary trees going into and out from the root do not touch except at their leaf vertices (at the ends of the connectors) and at the root. Finally, delete all nodes of incoming and outgoing binary trees that are not used for going from a node in G or the root to another node in G or the root. Then we obtain a strongly-connected directed graph, and this is $G''(k)$. Note also that $G''(k)$ has bounded degree. See Figure 2 for an illustration of this process.

3.3 Calculations

Consider the quantity $f(v) = d^*(root, v) + \max_{v' \in V''(k)} d(v, v')$ for each vertex in $v \in V''(k)$. The firing time of the network is $\max_{v \in V''(k)} f(v)$ by Lemma 1. Our overall goal will be to determine this value.

First, we split the vertices in $V''(k)$ into four major subdivisions and consider $f(v)$ for v in each subdivision. (A brief spoiler: It will turn out that the maximum value of $f(v)$ will occur in the fourth subdivision we consider. We need to calculate the function values for each of the other subdivisions in order to be sure.)

1. All vertices in the lower two binary trees, the final set of edges linking up with the root. It is clear that $d^*(root, v) = d(root, v)$ for vertices in this subdivision (with the exception of the root itself). Within this subdivision, consider vertices v that are oriented by their edges away from the root such that $d(root, v) = i \leq \log_2 n$. Note that the vertex $v' \in V''(k)$ that maximizes $d(v, v')$ will be somewhere in G' . The “cheapest” way to get there will be to head back to the root because the expense of jumping from one vertex to another in the graph G' is $3\log_2 n$ and this is more than the jump back to the root and to another path. Thus, $d(v, v') = (\log_2 n - i) + 2\log_2 n + k + (3\log_2 n - 1) = 6\log_2 n + k - i - 1$ and $f(v) = i + d(v, v') = 6\log_2 n + k - 1$.

Now consider vertices v in the same subdivision that are oriented towards the root such that $d(root, v) = \log_2 n + i$. By the same arguments, the same bound applies and thus $f(v) = 6\log_2 n + k - 1$ as well. Finally, note that $f(root) = 6\log_2 n + k - 1$ as well. (These values will turn out to be smaller than other potential values of $f(v)$ for other subdivisions; hence we will be able to assume that v is not located in either binary tree and these calculations will not contribute to the final determination of the firing time.)

2. The set of vertices in the connectors. (Note that this set has nonempty intersection with V .) Via the following lemma, we will show that only vertices in V need to be examined to find the maximal f value. These vertices will therefore be separated into their own subdivision below.

Lemma 3 *Within the connector subdivision, the function f is nondecreasing with increasing distance from the root.*

Proof. Consider any vertex v within a connector that is not the furthest possible from the root. Let v_{next} be the vertex one unit further from the root. Consider $f(v)$. Let v' be a vertex satisfying the maximum in the definition of f . Then $d(v_{next}, v') \geq d(v, v') - 1$. Because $d^*(root, v_{next}) \geq d^*(root, v) + 1$, we have

$$f(v_{next}) = d^*(root, v_{next}) + \max_{v'' \in V''(k)} d(v_{next}, v'') \geq d^*(root, v) + 1 + d(v_{next}, v') \geq d^*(root, v) + 1 + d(v, v') - 1 = f(v)$$

□

By the above lemma, we can assume without loss of generality that when locating the maximum within the connector subdivision, v is the vertex on the connector furthest from the root. However, this reduces us to only searching vertices V in the original graph $G \subset G'$. This will be the next subdivision we consider.

3. The vertices in the original graph G , namely V . We claim that $\forall v \in V, \exists v' \in V' - V$ such that $f(v') \geq f(v)$. In fact, note that the vertex v' on the terminal of any input from v will satisfy this claim. We omit the details due to space constraints.

4. $V' - V$. Let v be a vertex that maximizes $f(v)$ within the set $V' - V$ (i.e. $f(v) = \max_{v' \in V' - V} f(v')$). We will assume throughout this section without loss of generality that $d(G, v) = i$ for some $i \geq 1$. Then $d^*(root, v) = \log_2 n + k + i$. Let v' satisfy the maximum in the definition of f . Our goal is to determine the various values that $f(v)$ might take depending on where v' winds up being. First, however, we need a preliminary lemma. For the remainder of the paper, let the quantity $d_r(v, v')$ (resp. $d_n(v, v')$) be the length of the shortest path from v to v' that does (resp. does not) pass through the root.

Lemma 4 *Assume $v \in V' - V$. Then the vertex v' that maximizes $d(v, v')$ cannot be in either of the bottom binary trees assuming $k \geq 3\log_2 n$ and $D \geq 2$.*

Proof. If $k \geq 3 \log_2 n$, then there exists a vertex v' in the connector located $2 \log_2 n$ above the bottom of the connector. Note that

$$d_r(v, v') = 3 \log_2 n - i + k + 2 \log_2 n + 2 \log_2 n = 7 \log_2 n + k - i$$

and

$$d_n(v, v') = 3 \log_2 n - i + 3D \log_2 n + (k - 2 \log_2 n) \geq 7 \log_2 n + k - i$$

Thus $d(v, v') = 7 \log_2 n + k - i$ whereas the furthest vertex from v located within either binary tree is a distance $3 \log_2 n - i + k + \log_2 n + \log_2 n - 1 = 5 \log_2 n + k - i - 1$. \square

If $v \in V' - V$, there are several cases to consider for the location of v' and the behavior of the shortest path from v to v' . By the above lemma (Lemma 4), we only need to consider v' that lie within G' or the connectors.

- Assume that $v \in V' - V$, $v' \in V'$, and the shortest path from v to v' passes through the root. Then $f(v) = \log_2 n + k + i + (3 \log_2 n - i) + k + \log_2 n + \log_2 n + k + j$ where $d(G, v') = j < 3 \log_2 n$. Maximizing over the possible values of v' yields a total value of $9 \log_2 n + 3k - 1$.
- Assume that $v \in V' - V$, $v' \in V'$, and the shortest path from v to v' does not pass through the root. Then the path from v to v' must remain entirely inside G' . $f(v) = \log_2 n + k + i + (3 \log_2 n - i) + 3D \log_2 n + j$ where $d(G, v') = j$. Once again maximizing over the values of v' , we get $f(v) = (7 + 3D) \log_2 n + k - 1$ in this case.

Remark: It is worth noting at this point that by examination of the calculations above, $\exists v \in V' - V, \exists v' \in V'$ such that

$$f(v) = \min\{d^*(root, v) + d_n(v, v'), d^*(root, v) + d_r(v, v')\} = \min\{(7 + 3D) \log_2 n + k - 1, 9 \log_2 n + 3k - 1\}$$

In fact, we can pinpoint two vertices that fit the bill: Assume that two vertices a and b in G are the initial and terminal vertices of a path that equals the diameter of G . In $G''(k)$, consider the equivalent vertices. If we let v be anywhere on an incoming binary tree to a (not a itself) and we let v' be a furthest vertex from b on its outgoing binary tree, then the pair v and v' will satisfy the equalities above.

- Assume that $v \in V' - V$, $v' \notin V'$, and the shortest path from v to v' passes through the root. Note that v' is therefore in a connector. In this case, $f(v) = \log_2 n + k + 3 \log_2 n + k + \log_2 n + d(root, v')$. We also have that $d(root, v') = \log_2 n + k - d(v', G)$. So finally we get $f(v) = 6 \log_2 n + 3k - d(v', G)$ in this case.
- Assume that $v \in V' - V$, $v' \notin V'$, and the shortest path from v to v' does not pass through the root. Again, v' is in a connector. In this case, $f(v) = \log_2 n + k + 3 \log_2 n + 3D \log_2 n + d(G, v') = (4 + 3D) \log_2 n + k + d(G, v')$.

Note that each of these values is larger than the values calculated in any of the other subdivisions. Thus, we can conclude that *the function $f(v)$ is maximized just when $v \in V' - V$* . For the remainder of the paper, we will concentrate exclusively on this case.

Lemma 5 *Assume that $v \in V' - V$, and $v' \notin V'$. A vertex v' that maximizes $d(v, v')$ will be such that*

$$|d_n(v, v') - d_r(v, v')| \leq 1$$

Proof. By Lemma 4, v' is within a connector. Consider any v' within a connector such that $|d_n(v, v') - d_r(v, v')| \geq 2$. Assume without loss of generality that $d_n(v, v') > d_r(v, v')$. (The other case is symmetric.) Then $d_n(v, v') \geq d_r(v, v') + 2$. Consider the vertex v'' that is one “higher” than v' (i.e. the vertex that is one unit closer to G'). Then we have that $d_r(v, v'') = d_r(v, v') + 1$ and

$$d_n(v, v'') = d_n(v, v') - 1 \geq d_r(v, v') + 1 \Rightarrow d(v, v'') = d_r(v, v'') + 1$$

Because $d_n(v, v') > d_r(v, v')$, we have that $d(v, v') = d_r(v, v') < d(v, v'')$. \square

Lemma 6 *For some $v \in V' - V$ and some value of k . Assume that $v' \in V'$ is the vertex that maximizes the value of $d(v, v')$. If the shortest path from v to v' does not pass through the root, then if the value of k is increased, for a fixed v , as long as $v' \in V'$, the shortest path from v to v' will not pass through the root.*

Proof. Figure 2. Assume $d_n(v, v') < d_r(v, v')$. Increasing k can only increase the value of $d_r(v, v')$ but does not affect the value of $d_n(v, v')$. \square

Lemma 7 *Assume that $v \in V' - V$ and $v' \notin V'$ for some value of k and that v' is the vertex that maximizes the value of $d(v, v')$ (and is therefore in a connector by Lemma 4). Then with increasing values of k , v' will remain in the connector subdivision.*

Proof. Note that if $v' \notin V'$ then the distance from v to v' is greater than from v to any vertex in V' . Increasing the value of k only increases the distances of vertices outside of V' . \square

Lemma 8 *Assume that v is the vertex that maximizes the function $f(v)$. (Then, by the above discussion, $v \in V' - V$.) Let v' be the vertex that maximizes the value of $d(v, v')$. If $k > 4D \log_2 n$ and $D \geq 2$, then $v' \notin V'$.*

Proof. Figure 2. If $v \in V' - V$, then an obvious upper bound on the distance from v to any other vertex in V' is $(3D + 2) \log_2 n$. \square

3.4 The Algorithm

We now outline the algorithm we use to determine D . First, we determine whether $D = 1$ by checking whether every vertex is connected to every other vertex. This takes time $O(V + E)$. Now, starting from $k = 3 \log_2 n$, we simulate the minimal-time solution on the network $G''(k)$ and note the firing time. Note that once we examine the firing times assuming the four possible locations of the vertex v , we conclude that the maximum value of $f(v)$ is achieved when v is in $V' - V$. Thus, for the remainder of this section, we will implicitly assume this. The firing time must satisfy one of the four values calculated above for this situation.

If the firing time is $9 \log_2 n + 3k - 1$, then double the value of k and note the firing time again. (In the unlikely case that this value happens to be equal to one of the other possible three values, another doubling of k will suffice to distinguish.) By Lemmas 6 and 7, eventually we will either have $v' \in V'$ and a running time of $(7 + 3D) \log_2 n + k - 1$ or $v' \notin V'$. Note that if $v' \in V'$ at this point, then we must have (by the Remark in Section 3.3)

$$(7 + 3D) \log_2 n + k - 1 \leq 9 \log_2 n + 3k - 1 \Rightarrow k \geq \frac{3D - 2}{2} \log_2 n$$

If we quadruple the value of k , we are therefore guaranteed that $k > 4D \log_2 n$. Thus, quadruple the value of k and again note the firing time. At this point, we must have $v' \notin V'$ by Lemma 8.

By Lemma 5, we know that v' satisfies the inequality $|d_n(v, v') - d_r(v, v')| \leq 1$. Therefore either prediction (a) $6 \log_2 n + 3k - d(v', G)$ or (b) $(4 + 3D) \log_2 n + k + d(G, v')$ for the running time is at most off by 1. Note that because $v' \notin V'$, v' must be in a connector and therefore $d(G, v') = d(v', G)$. Because we know the value of the running time, we can estimate the value of $d(G, v')$ to within 1 using (a). Using (b) and this value, we can then estimate the value of $(4 + 3D) \log_2 n$ to within 2. The range for $(4 + 3D) \log_2 n$ is therefore 5 at most. If $\log_2 n > 1$, then at most a single value of D will satisfy the inequalities.

3.5 Time Analysis

To analyze the running time of the algorithm, we note that for a given value of k , to simulate the minimal-solution on the network $V''(k)$ in the RAM model of computation requires time proportional to the product of the number edges and the firing time. The firing time for any given value of k is clearly $O(D \log_2 n)$ because $k = O(D \log_2 n)$. The number of edges in the graph has been increased from E to $O(E \log_2 n)$. Thus the total simulation time is $O(ED \log_2 D (\log_2 n)^2)$. \square

4 Conclusions and Open Problems

In this paper, we have given a constructive proof that if a minimal-time solution exists for a fundamental distributed computation primitive, synchronizing a general directed network of finite-state processors, then there must exist an extraordinarily fast $O(ED \log_2 D (\log_2 n)^2)$ algorithm in the RAM model of computation for exactly determining the diameter of a general directed graph. This result opens up a number of very promising areas of research, the most obvious of which seem to be the following.

- Is this the best possible correspondence? If a minimal-time solution for the FSSP on the general directed network topology does happen to exist, is $O(ED \log_2 D (\log_2 n)^2)$ the best we can possibly do? Note that practically no effort was made to improve the simulation of the FSSP solution.
- Are results possible in the other direction? To date, there have been no results (to the author's knowledge, at least) that relate fast algorithms in standard models of computation such as the RAM model or the Turing Machine model to protocols for intercommunicating finite-state automata. Such a result would be extremely interesting.
- The approximation angle looks like a very promising area of research. If there exists an "almost" minimal-time solution, does it say anything about the algorithm for the diameter problem?
- What other problems can be related to the FSSP on various other topologies? It seems as if other natural topologies might (probably, should) have similar "natural" correspondences like this one.

References

- [1] Aingworth, Chekuri, and Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1996.
- [2] R. Balzer. An 8-state minimal time solution to the firing squad synchronization problem. *Information and Control*, 10:22–42, 1967.
- [3] A. Berthiaume, T. Bittner, L. Perković, A. Settle, and J. Simon. Bounding the firing synchronization problem on a ring. *Theoretical Computer Science*, 320:213–228, 2004.
- [4] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9:251–280, 1990.
- [5] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press/McGraw-Hill, 1990.
- [6] K. Culik. Variations of the firing squad problem and applications. *Information Processing Letters*, 30:153–157, 1989.
- [7] S. Even, A. Litman, and P. Winkler. Computing with snakes in directed networks of automata. *J. Algorithms*, 24(1):158–170, 1997.

- [8] M.L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM J. of Computing*, 5:83–89, 1976.
- [9] D. Goldstein and K. Kobayashi. On the complexity of network synchronization, to appear in *SIAM J. of Computing*.
- [10] D. Goldstein and K. Kobayashi. On the complexity of network synchronization. *Algorithms and Computation, Proc. 15th International Symposium, ISAAC 2004, HongKong, China*, 5(3):289–308, 2004.
- [11] E. Goto. A minimal time solution of the firing squad problem. *Course Notes for Applied Mathematics 298, Harvard University*, pages 52–59, 1962.
- [12] A. Grasselli. Synchronization of cellular arrays: The firing squad problem in two dimensions. *Information and Control*, 28:113–124, 1975.
- [13] T. Jiang. The synchronization of nonuniform networks of finite automata. In *Proc. of the 30th Annual ACM Symp. on Foundations of Computer Science*, pages 376–381, 1989.
- [14] T. Jiang. The synchronization of nonuniform networks of finite automata. *Information and Computation*, 97:234–261, 1992.
- [15] David R. Karger, Daphne Koller, and Steven J. Phillips. Finding the hidden path: Time bounds for all-pairs shortest paths. In *IEEE Symposium on Foundations of Computer Science*, pages 560–568, 1991.
- [16] K. Kobayashi. The firing squad synchronization problem for a class of polyautomata networks. *J. Comput. System Sci.*, 17(3):300–318, 1978.
- [17] K. Kobayashi. On the minimal firing time of the firing squad synchronization problem for polyautomata networks. *Theoretical Computer Science*, 7:149–167, 1978.
- [18] K. Kobayashi. On time optimal solutions of the firing squad synchronization problem for two-dimensional paths. *Theoretical Computer Science*, 259:129–143, 28 May 2001.
- [19] S. LaTorre, M. Napoli, and M. Parente. Synchronization of one-way connected processors. *Complex Systems*, 10:239–255, 1996.
- [20] J. Mazoyer. An overview of the firing synchronization problem. In *Automata Networks, LITP Spring School on Theoretical Computer Science, Angelès-Village, France, May 12-16, 1986, Proceedings*, volume 316 of *Lecture Notes in Computer Science*. Springer, 1988.
- [21] J. Mazoyer. Synchronization of two interacting finite automata. *International Journal of Algebra and Computation*, 5(3):289–308, 1995.
- [22] M. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Englewood Cliffs, NJ, 1967.
- [23] E. F. Moore. *Sequential Machines, Selected Papers*. Addison Wesley, Reading, MA, 1962.
- [24] F. R. Moore and G. G. Langdon. A generalized firing squad problem. *Information and Control*, 12:212–220, 1968.
- [25] Y. Nishitani and N. Honda. The firing squad synchronization problem for graphs. *Theoretical Computer Science*, 14(1):39–61, 1981.
- [26] R. Ostrovsky and D. Wilkerson. Faster computation on directed networks of automata. In *Proceedings of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 38–46, Ottawa, Ontario, Canada, 2–23 August 1995.
- [27] H. Schmid and T. Worsch. The firing squad synchronization problem with many generals for one-dimensional CA. *3rd IFIP International Conference on Theoretical Computer Science*, pages 111–124, 2004.
- [28] I. Shinahr. Two- and three-dimensional firing-squad synchronization problem. *Information and Control*, 24:163–180, 1974.
- [29] A. Waksman. An optimum solution to the firing squad synchronization problem. *Information and Control*, 9:66–78, 1966.
- [30] Uri Zwick. Exact and approximate distances in graphs.
- [31] Uri Zwick. A slightly improved sub-cubic algorithm for the all pairs shortest paths problem with real edge lengths. In Rudolf Fleischer and Gerhard Trippen, editors, *ISAAC*, volume 3341 of *Lecture Notes in Computer Science*, pages 921–932. Springer, 2004.