A scalable distributed dynamical systems approach to learn the strongly connected components and diameter of networks

Dedicated to Professors Cristina and Amilcar Sernadas

Emily A. Reed*, Guilherme Ramos*, Paul Bogdan, Sérgio Pequito

***Abstract*— Finding strongly connected components (SCCs) and the diameter of a directed network play a key role in a variety of machine learning and control theory problems. In this paper, we provide for the first time a scalable distributed solution for these two problems by leveraging dynamical consensus-like protocols to find the SCCs. The proposed solution has a time complexity of $\mathcal{O}\left(\mathsf{N}\mathsf{D}\mathrm{d}^{\max}_{\textbf{in-degree}}\right)$, where N is the number of vertices in the network, D is the (finite) diameter of the network and $\mathrm{d}^{\max}_{\textbf{in-degree}}$ is the maximum in-degree of the network. Additionally, we prove that our algorithm terminates in $\mathsf{D}+2$ iterations, which allows us to retrieve the finite diameter of the network. We perform exhaustive simulations that support the outperformance of our algorithm against the state-of-the-art on several random networks, including Erdős-Rényi, Barabási-Albert, and Watts-Strogatz networks.**

***Index Terms*— strongly connected components, max-consensus**

I. Introduction

Strongly connected components (SCCs) and the finite diameter of directed networks are important in many control theory problems. Nowadays, networks associated with data are becoming increasingly larger, which demands *scalable* and *distributed* algorithms that enable an efficient determination of the SCCs and diameter of such networks.

The applications of distributively finding the strongly connected components include distributively monitoring and regulating power systems, physiological networks, and swarms of unmanned vehicles. For example, these systems are often represented as structural systems [1] and are distributively controlled [2]. More specifically, strongly connected components are important in determining the structural systems properties (e.g., controllability and observability) [1] and play a key role in guaranteeing that distributed control algorithms function properly [2]. In the wide range of large-scale applications mentioned above, it is common that only local information is available at each node, and therefore, distributed algorithms, like the one proposed hereafter, must be considered.

Computing the finite diameter is important in improving internet search engines [3], quantifying the multifractal geometry of complex networks [4], and identifying faults in both the power grid [5] and multiprocessor systems [6]. More specifically, the finite diameter of the world wide web determines the maximum number of clicks between any two web pages [3]. Finding this number in a distributed fashion is important when there are multiple processors in a computer that are conducting different searches at the same time. In the case of identifying faults in systems such as the power grid [5] or multiprocessor systems [6], the finite diameter of a system is calculated in real-time and is compared with the known finite diameter to determine whether a fault has occurred. In this case, determining the finite diameter distributively is key in quickly diagnosing where the fault has occurred in the network. Finally, the finite diameter is important in quantifying the multifractal geometry of networks [4], and it becomes necessary to calculate the finite diameter distributively when the nodes of the network only have access to local information.

Identifying the different SCCs in a directed network (directed graph – digraph for short) leads to a unique decomposition of the digraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ denotes the nodes and $\mathcal{E}$ the set of directed edges. We may find this decomposition, for instance, using the classic algorithm by Tarjan [7], which employs a single pass of depth-first search and whose computational complexity is $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$. It is worth mentioning that depending on the network sparsity, the effective computational complexity is $\mathcal{O}(|\mathcal{V}|^2)$, since $\mathcal{E} \subset (\mathcal{V} \times \mathcal{V})$. Similar to Tarjan's algorithm, Dijkstra introduced the path-based algorithm to find strongly connected components and also runs in linear time (i.e., $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$) [8]. Finally, Kosaraju's algorithm uses two passes of depth-first search but is also upper-bounded by $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$ [9].

The following work presents an overview of the centralized algorithms that find the strongly connected components, which all have computational complexity $\mathcal{O}(|\mathcal{V}|+|\mathcal{E}|)$ [10]. A possible alternative is to develop better data structure algorithms that are suitable for parallelization, which can then lead to implementations with computational complexity equal to $\mathcal{O}(|\mathcal{V}| \log(|\mathcal{V}|))$ [11] – see also [12] for an overview of different parallelized algorithms for SCC decomposition.

The solutions mentioned above require the knowledge of the overall structure of the system digraph, which may not be suitable for large-scale applications in control systems or in machine learning. Subsequently, we propose for the first time a scalable distributed algorithm to determine the SCCs that relies solely on control systems tools, specifically max-consensus-like dynamics. Furthermore, our algorithm converges in $D + 2$ iterations and thereby enables us to determine the finite diameter D of the network. State-of-the-art methods to determine the finite diameter of a directed network include the Floyd-Warshall algorithm, which has a computational complexity of $\mathcal{O}(|\mathcal{V}|^3)$ [13]. The main contributions of the paper are as follows.

Main contributions:

- We provide for the first time a scalable distributed algorithm to find the strongly connected components and finite diameter of a directed graph with computational time-complexity $\mathcal{O}\left(ND\mathrm{d}^{\max}_{\text{in-degree}}\right)$, and
- we provide ample numerical evidence of the out performance of our algorithm against the state-of-the-art on several random networks including Erdős-Rényi, Barabási-Albert, and Watts-Strogatz.

A. Preliminaries and Terminology

Consider a directed graph (digraph) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of vertices with $|\mathcal{V}| = N$, and $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ is the set of edges, where the maximum number of edges is $|\mathcal{E}| = |\mathcal{V} \times \mathcal{V}| = N^2$. Given $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the *in-degree* of a vertex $v \in \mathcal{V}$ is $d_{\text{in-degree}}(v) = |\{(u, u') : (u, u') \in \mathcal{E}, u' = v\}|$, and we denote the maximum in-degree of $\mathcal{G}$ by $d^{\max}_{\text{in-degree}} = \max_{v \in \mathcal{V}} d_{\text{in-degree}}(v)$. Moreover, given a vertex $v \in \mathcal{V}$, we define the set of its *in-neighbors* as $\mathcal{N}_v^- = \{u : (u, v) \in \mathcal{E}\}$.

* Both authors contributed equally.
E. Reed is with the Ming Hsieh Electrical and Computer Engineering Dept. at the University of Southern California, USA. G. Ramos is an assistant professor at the Department of Computer Science Engineering, Instituto Superior Técnico, Universidade de Lisboa and an integrated researcher at LASIGE, Departamento de Informática, Faculdade de Ciências, Universidade de Lisboa. P. Bogdan is a faculty member at the University of Southern California in the Ming Hsieh Electrical and Computer Engineering Dept. S. Pequito is an associate professor of automatic control in the Department of Information Technology at Uppsala University. This work was supported in part by FCT through the LASIGE Research Unit ref. UIDB/00408/2020 and ref. UIDP/00408/2020, National Science Foundation GRFP DGE-1842487, Career Award CPS/CNS-1453860, CCF-1837131, MCB-1936775, CNS-1932620, CMMI-1936624, CMMI 1936578, the University of Southern California Annenberg Fellowship, USC WiSE Top-Off Fellowship, the DARPA Young Faculty Award and DARPA Director Award N66001-17-1-4044. The views, opinions, and/or findings contained in this article are those of the authors and should not be interpreted as representing the official views or policies, either expressed or implied by the Defense Advanced Research Projects Agency, the Dept. of Defense or the National Science Foundation.

A *walk* in a digraph is any sequence of edges where the last vertex in one edge is the beginning of the next edge, except for the beginning vertex of the first edge and the ending vertex of the last edge. Notice that a walk does not exclude the repetition of vertices. In contrast, a *path*, is a walk where the same vertex is not the beginning or ending of two nonconsecutive edges in the sequence. The size of the path is the number of edges that constitute it. If the beginning and ending vertex of a path is the same, then we obtain a *cycle*. Additionally, a *sub-digraph* $\mathcal{G}_s = (\mathcal{V}', \mathcal{E}')$ is described as any sub-collection of vertices $\mathcal{V}' \subset \mathcal{V}$ and the edges $\mathcal{E}' \subset \mathcal{V}' \times \mathcal{V}'$ between them. If a subgraph has the property that there exists a path between any two pairs of vertices, then it is a *strongly connected (di)graph*. Next, we provide the definition of a strongly connected component.

Definition 1. *A* strongly connected component (SCC) *is any maximal strongly connected subgraph.*

Any digraph can be uniquely decomposed into SCCs. A digraph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ is said to *span* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, denoted by $\mathcal{G}' = \text{span}(\mathcal{G})$, if $\mathcal{V}' = \mathcal{V}$ and $\mathcal{E}' \subseteq \mathcal{E}$. Finally, given a digraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we define its finite digraph diameter D.

Definition 2. *The* finite digraph diameter *is the size of the longest shortest path between any two vertices in $\mathcal{V}$ for which such a path exists.*

II. Problem Statement

We propose to address the following two problems.

$(\mathbf{P}_1)$: Given a digraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, determine the unique decomposition of $m \in \mathbb{N}$ SCCs by finding the maximal subgraphs $\mathcal{G}_s = (\mathcal{V}_s, \mathcal{E}_s), s = 1, \cdots, m$, where each subgraph is a SCC such that $\mathcal{V}_s \cap \mathcal{V}_q = \emptyset$ for $s \neq q$ with $q = 1, \ldots, m$, $\mathcal{V}_s, \mathcal{V}_q \subset \mathcal{V}$, $\mathcal{E}_s \subset (\mathcal{E} \cap (\mathcal{V}_s \times \mathcal{V}_s))$, and $\cup_{s=1}^{m} \mathcal{G}_s \equiv (\cup_{s=1}^{m} \mathcal{V}_s, \cup_{s=1}^{m} \mathcal{E}_s) = \text{span}(\mathcal{G})$.

$(\mathbf{P}_2)$: Given a digraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, determine the finite digraph diameter D.

Next, we provide the solution to the above problems in both a centralized and distributed fashion that enables a scalable approach to determine all the SCCs and the finite digraph diameter of a given network. Notice that a digraph has a unique decomposition into m SCCs, but we do not require *a priori* knowledge of such number.

III. A Scalable Distributed Dynamical Systems Approach to Learn the Strongly Connected Components and Diameter of Networks

To determine a solution to $(\mathbf{P}_1)$ and $(\mathbf{P}_2)$, we leverage a max-consensus-like protocol.

Definition 3. *[14] Consider $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where each vertex $v_i \in \mathcal{V}, i = 1, \ldots, N$, has an associated state $r_i[k] \in \mathbb{R}$ at any time $k \in \mathbb{N}$. Then, we have the following* max-consensus-like *update rule*

$$r_i[k+1] = \max_{v_j \in \mathcal{N}_{v_i}^- \cup \{v_i\}} r_j[k], \tag{1}$$

for each node v_i, where $\mathcal{N}_{v_i}^-$ denotes all of the nodes v_j such that there is an edge $(v_j, v_i) \in \mathcal{E}$. We simply say that consensus is achieved if there exists an instance of time h such that for all $h' \geq h$, $r_i[h'] = r_j[h']$, for all $v_i, v_j \in \mathcal{V}, i \in \{1, \ldots, N\}, j \in \{1, \ldots, N\}$ and for all initial conditions $r[0] = [r_1[0]^\intercal \ldots r_n[0]^\intercal]^\intercal$. ∘

Definition 3 is similar to the max-consensus update, but in addition to considering the information from the neighbors, Definition 3 also considers the information from the node itself. Furthermore, it is worth emphasizing that from Definition 3 it follows that every node only needs to be able to receive information from its in-neighbors, (i.e., the nodes connected to it). Hence, each node only needs the local information, which is pertinent to distributed algorithms. Furthermore, we note that each node uses a unique ID, that for simplicity consists of N consecutive positive integer numbers from 1 to N. A node is able to obtain the information from its in-neighbors, including the unique IDs of its in-neighbors.

Algorithm 1: Find the SCCs and finite diameter distributively

Input: $\mathcal{N}_{v_i}^-$, which is the set of in-neighbors of node v_i
Output: Each node v_i obtains a set $\mathcal{S}_i^*$, which contains the nodes belonging to the same SCC as node v_i, and a scalar k_i, which is one more than the number of iterations
Initialization: Set $\mathcal{S}_i^* = \emptyset$, $k_i = 0$; $x_i[0] = \{i\}$; $y_i[0] = 1$; $z_i[0] = \emptyset$; and $w_i[0] = \text{FALSE}$;
while $w_i[k_i] == \text{FALSE}$ **do**

Step 1:

$$x_i[k_i+1] = \bigcup_{v_j \in \mathcal{N}_{v_i}^- \cup \{v_i\}} x_j[k_i]$$

Step 2:

$$y_i[k_i+1] = |x_i[k_i+1]|$$

Step 3:

$$z_i[k_i+1] = \left\{ v_j \,:\, y_i[k_i+1] == y_j[k_i] \wedge v_j \in \bigcup_{v_l \in \mathcal{N}_{v_i}^- \cup \{v_i\}} x_l[k_i] \right\}$$

Step 4:

$$w_i[k_i+1] = (y_i[k_i+1] == y_i[k_i]) \wedge (|z_i[k_i+1]| == |z_i[k_i]|)$$

Step 5: $k_i = k_i + 1$
end
Step 6: Set $S_i^* = z_i[k_i]$

Next, we present Algorithm 1, which can be used to find the solutions to $(\mathbf{P}_1)$ and $(\mathbf{P}_2)$.

Algorithm 1 is performed on each node v_i and obtains a set $\mathcal{S}_i^*$, which consists of the nodes that belong to the same SCC as node v_i, and a scalar k_i, which is one more than the number of iterations.

Briefly speaking, Algorithm 1 works as follows. For each node v_i, we first find the set of nodes that have a directed path ending in node v_i. Next, we record the size of this set. Finally, we add the nodes contained in the same SCC as node v_i to the set $\mathcal{S}_i^*$.

More specifically, Algorithm 1 starts by initializing the local (i.e., at node v_i) sets and parameters for the algorithm. In particular, we set $\mathcal{S}_i^* = \emptyset$, $k_i = 0$, $x_i[0] = \{i\}$, $y_i[0] = 1$, $z_i[0] = \emptyset$, and $w_i[0] = \text{FALSE}$. At each iteration of the algorithm, Step 1 finds the set of state 'ids' (or, equivalently, nodes' indices) that form directed paths that end in node v_i. Step 2 records the size of the set of directed paths to node v_i. Step 3 determines the nodes that are contained in the same SCC as node v_i, where $\wedge$ denotes the logical 'and' operation. In Step 4, if the maximum size of the set of directed paths to v_i has been obtained, then an indication to end the algorithm for node v_i is provided. Step 5 tracks the iterations, which is important for finding the finite digraph diameter. The algorithm terminates when no new information is received. Lastly, Step 6 sets $\mathcal{S}_i^*$, which is the set of nodes contained in the same SCC as node v_i.

The following lemma is key in proving Algorithm 1's correctness.

Lemma 1. *If, for any two nodes v_i and v_j, we have that $y_i[k_i+1] = y_j[k_i]$ (as in Step 2) and $v_j \in \bigcup_{v_l \in \mathcal{N}_{v_i}^- \cup \{v_i\}} x_l[k_i]$ (as in Step 1), then v_i and v_j are in the same SCC.*

Proof. Suppose for a contradiction that $y_i[k_i+1] = y_j[k_i]$ and $v_j \in \bigcup_{v_l \in \mathcal{N}_{v_i}^- \cup \{v_i\}} x_l[k_i]$, but v_i and v_j are not in the same SCC. This would mean that there is not a direct path from v_i to v_j or there is not one from v_j to v_i. However, if $v_j \in \cup_{v_l \in \mathcal{N}_i^- \cup \{i\}} x_l[k_i]$, then v_j can reach node v_i, so there is a direct path from v_j to v_i. Furthermore, if $y_i[k_i+1] = y_j[k_i]$, then there must also be a direct path from v_i to v_j or we would have $y_i[k_i+1] > y_j[k_i]$. Therefore, there is a direct path from v_i to v_j and from v_j to v_i, so v_i and v_j must be in the same SCC. □

The next lemma is key in giving a stopping criteria for Algorithm 1.

Lemma 2. *If $y_i[k_i+1] == y_i[k_i]$ and $|z_i[k_i+1]| == |z_i[k_i]|$, for all $i = 1, \ldots, N$ (as in Step 4), then all the SCCs have been found.*

Proof. At each iteration of the algorithm, the number of elements in set x_i either increases or it remains the same. If from one iteration to the next, the number of elements in x_i stays the same for all nodes v_i, then every node has received all of the information that it possibly can. Furthermore, each node knows the other nodes that can reach it as captured by the set x_i. Since y_i records the size of the set of nodes that can reach node v_i, then if y_i remains the same from one iteration to the next, then it is clear that the number of elements in x_i also remains the same. Furthermore, if y_i remains the same from one iteration to the next, then the network cannot communicate any new information. Additionally, if the set z_i, which is the set of nodes that are contained in the same SCC as node v_i, remains the same from one iteration to the next, then all SCCs have been found. □

In the following theorem, we prove the correctness of Algorithm 1.

Theorem 1. *Let $\mathcal{S}_i^*$ be the set of nodes that results after Algorithm 1 is executed on node $v_i \in \mathcal{V}$. Then, $\bigcup_{i\in\{1,\ldots,N\}}(\mathcal{S}_i^*, (\mathcal{S}_i^* \times \mathcal{S}_i^*) \cap \mathcal{E})$ is a solution to $\mathbf{P}_1$.* ∘

Proof. The algorithm iterates until $y_i[k_i+1] = y_i[k_i]$ and $|z_i[k_i+1]| = |z_i[k_i]|$, for all $i = 1, \ldots, N$, at which point all of the SCCs have been found–see Lemma 2. At each iteration, Step 1 forms the set of nodes that reach node v_i and is recorded in set x_i. Step 2 finds the cardinality of the set x_i. Step 3 finds the set of nodes that are contained in the same SCC as node v_i – see Lemma 1 – and is recorded in set z_i. Step 4 determines whether the maximum set of nodes that can reach node v_i has been found, whether the size of the set of nodes contained in the same SCC has increased, and it serves as a stopping criteria for the algorithm. Step 5 tracks the iterations, and step 6 records the set of nodes that are contained in the same SCC as node v_i in the set $\mathcal{S}_i^*$. Finally, the SCCs are formed in the following subgraphs $\mathcal{G}_i^* = (\mathcal{S}_i^*, (\mathcal{S}_i^* \times \mathcal{S}_i^*) \cap \mathcal{E})$ as mentioned in the statement of Theorem 1. Any duplicate subgraphs of SCCs are eliminated by taking the union of all the subgraphs $\bigcup_{i\in\{1,\ldots,N\}} \mathcal{G}_i^*$. Hence, we obtain the SCCs of $\mathcal{G}(\mathcal{V}, \mathcal{E})$. □

Next, we show the computational time-complexity for Algorithm 1.

Theorem 2. *Algorithm 1 has computational time-complexity $\mathcal{O}\left(N^2 D d^{\max}_{\textit{in-degree}}\right)$, where N is the number of vertices, D is the finite diameter of the network and $d^{\max}_{\textit{in-degree}}$ is the maximum in-degree of the network.* ∘

Proof. Algorithm 1 executes for all nodes $v_i \in \mathcal{V}, i = 1, \ldots, N$. Furthermore, Algorithm 1 contains a single while loop, where the number of iterations is upper-bounded by the diameter since x_i finds the longest shortest path to node v_i. The steps inside the while loop are steps 1-5. Step 1 is upper-bounded by the size of the network times the maximum in-degree of the network (i.e., $\mathcal{O}(N d^{\max}_{\text{in-degree}})$) since the union has complexity $\mathcal{O}(N)$ and we take the union for all of the in-neighbors. Step 2 is upper-bounded by a constant. Similar to Step 1, Step 3 is upper-bounded $\mathcal{O}(N d^{\max}_{\text{in-degree}})$. Finally, steps 4, 5, and 6 are upper-bounded by a constant. Hence, the computational time-complexity is $\mathcal{O}\left(N^2 D d^{\max}_{\text{in-degree}}\right)$, where N is the number of nodes, D is the finite digraph diameter of the network, and $d^{\max}_{\text{in-degree}}$ is the maximum in-degree of the network. □

The following result demonstrates the scability of Algorithm 1.

Corollary 1. *Algorithm 1 can be implemented in a distributed fashion and is scalable with computational time-complexity $\mathcal{O}\left(N D d^{\max}_{\textit{in-degree}}\right)$.* ∘

Proof. This readily follows from Theorem 2 and from noticing that Steps 1-6 can be performed locally for each node v_i, where $i = 1, \ldots, N$. Therefore, the algorithm can be computed in a distributed fashion, which eliminates an N in the complexity in Theorem 2. □

While we have provided the computational complexity of the algorithm for the worst-case scenario, we observe that intuitively the maximum in-degree is negatively correlated with the diameter, so (on average) this may allow for a lower time complexity.

The space-complexity for performing Algorithm 1 on each node v_i, where $i \in \{1, \ldots, N\}$, in a distributed fashion is $\mathcal{O}(N)$, where N is the number of nodes in the network. Next, we give the computational time-complexity to find the SCCs.

Theorem 3. *Finding the sets of nodes that contain the SCCs, $\bigcup_{i\in\{1,\ldots,N\}}\mathcal{S}_i^*$, requires a computational time-complexity of $\mathcal{O}\left(N D d^{\max}_{\textit{in-degree}}\right)$.*

Proof. As shown in Theorem 1, for each node v_i, where $i = 1, \ldots, N$, Algorithm 1 finds the set of nodes $\mathcal{S}_i^*$ that are contained in the same SCC as node v_i, which, according to Corollary 1, has a computational time-complexity of $\mathcal{O}\left(N D d^{\max}_{\text{in-degree}}\right)$ when executed distributively. To find the sets of nodes that contain the SCCs, we compute $\bigcup_{i\in\{1,\ldots,N\}}\mathcal{S}_i^*$, which requires the addition of a term N, so $\mathcal{O}\left(N D d^{\max}_{\text{in-degree}} + N\right)$. Thus, it readily leads to $\mathcal{O}\left(N D d^{\max}_{\text{in-degree}}\right)$. □

Next, we provide a table comparing the computational time complexities of several different algorithms that find the SCCs of a given directed network.

Algorithm to compute SCCs	Computational Time Complexity
Tarjan [7]	$\mathcal{O}(N + \lvert\mathcal{E}\rvert)$
Dijkstra [8]	$\mathcal{O}(N + \lvert\mathcal{E}\rvert)$
Kosaraju [9]	$\mathcal{O}(N + \lvert\mathcal{E}\rvert)$
Gabow [15]	$\mathcal{O}(N + \lvert\mathcal{E}\rvert)$
Our proposed distributed algorithm	$\mathcal{O}(N D d^{\max}_{\text{in-degree}})$

While asymptotically the computational time-complexity of our distributed algorithm does not outperform the state-of-the-art centralized algorithms that find the SCCs (since the number of edges $|\mathcal{E}|$ in the graph is bounded by $N \times d^{\max}_{\text{in-degree}}$), our proposed algorithm can be implemented in a distributed manner, whereas the other algorithms shown in the table above cannot be. Furthermore, as we will show in the simulation results, our centralized algorithm empirically outperforms Kosaraju's algorithm on several randomly generated networks. In the next result, we give a solution to $(\mathbf{P}_2)$.

Theorem 4. *By running Algorithm 1 on every node $v_i \in \mathcal{V}$, with $i \in \{1, \ldots, N\}$, we get the solution to $(\mathbf{P}_2)$ to be $D = \max_{v_i\in\mathcal{V}} k_i - 3$.* ∘

Proof. We will show that Algorithm 1 converges after $D + 2$ iterations, where D is the finite digraph diameter of the input digraph. From Lemma 1, the algorithm terminates when no new information is being received by any node from its neighbors (or itself) at a subsequent time step and all of the SCCs have been found. If we assume that the digraph has diameter D, this implies that there exists a pair of nodes u and v such that the size of the shortest path between u and v is D. First, we will show that no new information is received in D iterations. Suppose that node v receives all the information of node u in $k < D$ iterations where the information travels to the neighbors of each node in exactly one iteration. Then, there must be another path from u to v with k edges, which contradicts the fact that the shortest path between u and v has size D.

Now, suppose that no new information is communicated to any node in the network after $k > D$ iterations. This means that there is information from node u that only reaches node v after k iterations. However, since information is sent to the neighbors at each iteration, then the shortest path between u and v has size k, which contradicts the fact that the longest shortest finite path has size D.

Therefore, no new information is being communicated in D iterations, which is verified in the next iteration, i.e., the $D + 1$th iteration. Then, z_i is finished updating after $D+1$ iterations since it is dependent on all of the information having been received. It is verified that z_i is finished updating at the next iteration, i.e., the $D + 2$nd iteration. Hence, the diameter will be two less than the maximum

number of iterations among all nodes. Since Step 5 increments k_i before terminating, then, k_i denotes the number of iterations (for v_i) plus one. Conveniently, $D = \max_{v_i \in \mathcal{V}} k_i - 3$ since $\max_{v_i \in \mathcal{V}} k_i$ finds precisely the maximum number of iterations plus one among all nodes v_i, so three is subtracted to obtain the finite digraph diameter D. □

We emphasize that computing the finite digraph diameter requires the number of iterations of Algorithm 1 for each node. Furthermore, we notice that we can compute the finite digraph diameter and the digraph diameter. For example, if there is more than one SCCs, then the digraph diameter is infinite, else the two quantities are equivalent. Next, we give the computational time-complexity for computing the finite digraph diameter.

Theorem 5. *Computing the finite digraph diameter requires a computational time-complexity of* $\mathcal{O}\left(ND d^{\max}_{\textit{in-degree}}\right)$.

Proof. Following from Theorem 4, we obtain the finite digraph diameter by executing Algorithm 1 on every single node $v_i \in \mathcal{V}$. Hence, from Corollary 1, we see that Algorithm 1 has a computational time-complexity of $\mathcal{O}\left(ND d^{\max}_{\text{in-degree}}\right)$ when executed distributively. To determine the maximum number of iterations among all of the nodes v_i, where $i \in 1, \ldots, N$, a final term N is added in the complexity, so $\mathcal{O}\left(ND d^{\max}_{\text{in-degree}} + N\right)$. Thus, it readily leads to $\mathcal{O}\left(ND d^{\max}_{\text{in-degree}}\right)$. □

We provide a table comparing the computational-time complexity of the state-of-the-art Floyd-Warshall algorithm to our proposed distributed algorithm. We notice that our proposed distributed algorithm performs no worse than the Floyd-Warshall algorithm.

Algorithm to find the finite diameter	Computational Time Complexity
Floyd-Warshall [13]	$\mathcal{O}(N^3)$
Our proposed distributed algorithm	$\mathcal{O}(ND d^{\max}_{\text{in-degree}})$

Finally, we explore the expected computational time-complexity of Algorithm 1 in some special random networks.

Corollary 2. *For an Erdős–Rényi network with N nodes and m edges, the expected time-complexity of Algorithm 1 is* $\mathcal{O}\left(\left(\frac{\log(N)-\gamma}{\log(2m/N)} + \frac{1}{2}\right) 2m\right)$, *where γ is the Euler-Mascheroni constant.*

Proof. The average degree of an Erdős–Rényi network is $\frac{2m}{N}$, and the average path length is $\frac{\log(N)-\gamma}{\log(2m/N)} + \frac{1}{2}$ [16]. Thus, by Corollary 1, the expected time-complexity is $\mathcal{O}\left(\left(\frac{\log(N)-\gamma}{\log(2m/N)} + \frac{1}{2}\right) 2m\right)$, where γ is the Euler-Mascheroni constant. □

Corollary 3. *For a Barabási-Albert network with N nodes and m edges added to a new vertex at each step, the expected time-complexity of Algorithm 1 is* $\mathcal{O}\left(2mN\left(\frac{\log(N)-\log(m/2)-1-\gamma}{\log(\log(N))+\log(m/2)} + \frac{3}{2}\right)\right)$.

Proof. Since the average degree is $2m$, and the average path length is $\frac{\log(N)-\log(m/2)-1-\gamma}{\log(\log(N))+\log(m/2)} + \frac{3}{2}$ [16], by Corollary 1, the average time-complexity reduces to $\mathcal{O}\left(2mN\left(\frac{\log(N)-\log(m/2)-1-\gamma}{\log(\log(N))+\log(m/2)} + \frac{3}{2}\right)\right)$. □

Corollary 4. *For a Watts-Strogatz network with N nodes, K edges per vertex, and rewiring probability p, the expected time-complexity of Algorithm 1 is* $\mathcal{O}\left(\frac{N^2}{2}\right)$ *as* $p \to 0$ *and* $\mathcal{O}\left(\frac{NK\log(N)}{\log(K)}\right)$ *as* $p \to 1$.

Proof. The Watts-Strogatz network has an average degree of K, and the average path length is $\frac{N}{2K}$ as $p \to 0$ and $\frac{\log(N)}{\log(K)}$ as $p \to 1$ [17]. Hence, by Corollary 1, the average time-complexity of Algorithm 1 for the Watts-Strogatz network is $\mathcal{O}\left(\frac{N^2}{2}\right)$ as $p \to 0$ and $\mathcal{O}\left(\frac{NK\log(N)}{\log(K)}\right)$ as $p \to 1$. □

IV. Pedagogical Examples

In this section, we present several pedagogical examples to illustrate how Algorithm 1 works and demonstrate its computational time-complexity. In what follows, when referring to each of the SCCs, we will only mention the indices of the nodes contained in that particular SCC (i.e., if $v_i \in \mathcal{V}_s$ then with some abuse of notation we refer to that node as $i \in \mathcal{V}_s$) as we are implicitly assuming that their edges are formed by $\mathcal{E}_s = ((\mathcal{V}_s \times \mathcal{V}_s) \cap \mathcal{E})$.

A. Example 1

Fig. 1 shows a network with six nodes that contains the following strongly connected components: $\{1, 2\}, \{3, 4\}$, and $\{5, 6\}$.

Fig. 1: This network has SCCs $\{5, 6\}, \{3, 4\}$, and $\{1, 2\}$.

Table I shows the trace of running Algorithm 1 on Example 1 for each node v_i, where $\mathcal{P}$ is the set of parameters for the algorithm and k is the total number of iterations. It shows in column one that it takes seven iterations ($k = 7$) to identify the SCCs for Example 1. Here, the diameter of the network is 5, which is two less than the total required iterations and is consistent with the results in Theorem 4.

k	$\mathcal{P}$	v_1	v_2	v_3	v_4	v_5	v_6
0	x[0]	{1}	{2}	{3}	{4}	{5}	{6}
	y[0]	1	1	1	1	1	1
	z[0]	{1}	{2}	{3}	{4}	{5}	{6}
	w[0]	False	False	False	False	False	False
1	x[1]	{1,2}	{1,2}	{2,3,4}	{3,4}	{4,5,6}	{5,6}
	y[1]	2	2	3	2	3	2
	z[1]	{}	{}	{}	{}	{}	{}
	w[1]	False	False	False	False	False	False
2	x[2]	{1,2}	{1,2}	{1,2,3,4}	{2,3,4}	{3,4,5,6}	{4,5,6}
	y[2]	2	2	4	3	4	3
	z[2]	{1,2}	{1,2}	{}	{3}	{}	{5}
	w[2]	False	False	False	False	False	False
3	x[3]	{1,2}	{1,2}	{1,2,3,4}	{1,2,3,4}	{2,3,4,5,6}	{3,4,5,6}
	y[3]	2	2	4	4	5	4
	z[3]	{1,2}	{1,2}	{3}	{3}	{}	{3,5}
	w[3]	True	True	False	False	False	False
4	x[4]	{1,2}	{1,2}	{1,2,3,4}	{1,2,3,4}	{1,2,3,4,5,6}	{2,3,4,5,6}
	y[4]	2	2	4	4	6	5
	z[4]	{1,2}	{1,2}	{3,4}	{3,4}	{}	{5}
	w[4]	True	True	False	False	False	False
5	x[5]	{1,2}	{1,2}	{1,2,3,4}	{1,2,3,4}	{1,2,3,4,5,6}	{1,2,3,4,5,6}
	y[5]	2	2	4	4	6	6
	z[5]	{1,2}	{1,2}	{3,4}	{3,4}	{5}	{5}
	w[5]	True	True	True	True	False	False
6	x[6]	{1,2}	{1,2}	{1,2,3,4}	{1,2,3,4}	{1,2,3,4,5,6}	{1,2,3,4,5,6}
	y[6]	2	2	4	4	6	6
	z[6]	{1,2}	{1,2}	{3,4}	{3,4}	{5,6}	{5,6}
	w[6]	True	True	True	True	False	False
7	x[7]	{1,2}	{1,2}	{1,2,3,4}	{1,2,3,4}	{1,2,3,4,5,6}	{1,2,3,4,5,6}
	y[7]	2	2	4	4	6	6
	z[7]	{1,2}	{1,2}	{3,4}	{3,4}	{5,6}	{5,6}
	w[7]	True	True	True	True	True	True

TABLE I: This table enumerates the values of the parameters ($\mathcal{P}$) at each iteration (k) of Algorithm 1 for all nodes v_i when executed on Example 1.

B. Example 2: Complete Network

Fig. 2 shows a complete network with five nodes, so there is a single SCC containing all of the nodes (i.e., $\{1, 2, 3, 4, 5\}$). In Table II, we see that three iterations are necessary as this is two more than the diameter of the network – see Theorem 4.

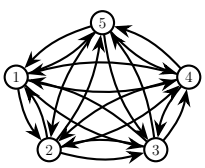

Fig. 2: The complete network contains a single SCC, which is made up of all of the nodes in the network (i.e., $\{1, 2, 3, 4, 5\}$).

C. Example 3: Tree

Fig. 3 shows a tree with nine nodes, so the SCCs are the individual nodes themselves (i.e., $\{1\}, \{2\}, \ldots, \{9\}$).

This article has been accepted for publication in IEEE Transactions on Automatic Control. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TAC.2022.3209446

k	$\mathcal{P}$	v_1	v_2	v_3	v_4	v_5
0	x[0]	{1}	{2}	{3}	{4}	{5}
	y[0]	1	1	1	1	1
	z[0]	{1}	{2}	{3}	{4}	{5}
	w[0]	False	False	False	False	False
1	x[1]	{1,2,3,4,5}	{1,2,3,4,5}	{1,2,3,4,5}	{1,2,3,4,5}	{1,2,3,4,5}
	y[1]	5	5	5	5	5
	z[1]	{}	{}	{}	{}	{}
	w[1]	False	False	False	False	False
2	x[2]	{1,2,3,4,5}	{1,2,3,4,5}	{1,2,3,4,5}	{1,2,3,4,5}	{1,2,3,4,5}
	y[2]	5	5	5	5	5
	z[2]	{1,2,3,4,5}	{1,2,3,4,5}	{1,2,3,4,5}	{1,2,3,4,5}	{1,2,3,4,5}
	w[2]	False	False	False	False	False
3	x[3]	{1,2,3,4,5}	{1,2,3,4,5}	{1,2,3,4,5}	{1,2,3,4,5}	{1,2,3,4,5}
	y[3]	5	5	5	5	5
	z[3]	{1,2,3,4,5}	{1,2,3,4,5}	{1,2,3,4,5}	{1,2,3,4,5}	{1,2,3,4,5}
	w[3]	True	True	True	True	True

TABLE II: This table enumerates the values of the parameters ($\mathcal{P}$) at each iteration (k) of Algorithm 1 for all nodes v_i when executed on the complete network.

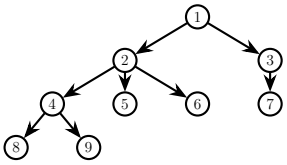

Fig. 3: The SCCs of the tree are the individual nodes themselves (i.e., $\{1\}, \{2\}, \ldots, \{9\}$).

k	$\mathcal{P}$	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
0	x[0]	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}	{9}
	y[0]	1	1	1	1	1	1	1	1	1
	z[0]	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}	{9}
	w[0]	False	False	False	False	False	False	False	False	False
1	x[1]	{1}	{1,2}	{1,3}	{2,4}	{2,5}	{2,6}	{3,7}	{4,8}	{4,9}
	y[1]	1	2	2	2	2	2	2	2	2
	z[1]	{1}	{}	{}	{}	{}	{}	{}	{}	{}
	w[1]	True	False	False	False	False	False	False	False	False
2	x[2]	{1}	{1,2}	{1,3}	{1,2,4}	{1,2,5}	{1,2,6}	{1,3,7}	{2,4,8}	{2,4,9}
	y[2]	1	2	2	3	3	3	3	3	3
	z[2]	{1}	{2}	{3}	{}	{}	{}	{}	{}	{}
	w[2]	True	False	False	False	False	False	False	False	False
3	x[3]	{1}	{1,2}	{1,3}	{1,2,4}	{1,2,5}	{1,2,6}	{1,3,7}	{1,2,4,8}	{1,2,4,9}
	y[3]	1	2	2	3	3	3	3	4	4
	z[3]	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{}	{}
	w[3]	True	True	True	False	False	False	False	False	False
4	x[4]	{1}	{1,2}	{1,3}	{1,2,4}	{1,2,5}	{1,2,6}	{1,3,7}	{1,2,4,8}	{1,2,4,9}
	y[4]	1	2	2	3	3	3	3	4	4
	z[4]	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}	{9}
	w[4]	True	True	True	True	True	True	True	False	False
5	x[5]	{1}	{1,2}	{1,3}	{1,2,4}	{1,2,5}	{1,2,6}	{1,3,7}	{1,2,4,8}	{1,2,4,9}
	y[5]	1	2	2	3	3	3	3	4	4
	z[5]	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}	{9}
	w[5]	True	True	True	True	True	True	True	True	True

TABLE III: This table shows the values of the parameters ($\mathcal{P}$) at each iteration (k) of Algorithm 1 for all nodes v_i, executed on the tree network.

In Table III, we see that five iterations are required to identify the SCCs of the tree network, which is two more than the diameter of the network – see Theorem 4.

V. Simulation Results

In this section, we compare the performance of our centralized algorithm with the current state-of-the-art that find the SCCs and the finite digraph diameter. We start by comparing the run times of our algorithm against Kosaraju's algorithm [9] to find all the SCCs on a series of random networks, including the Erdős–Rényi, Barabási–Albert, and Watts-Strogatz networks. We compare the run times of both our algorithm and the Kosaraju algorithm as we vary the parameters of the networks, including the diameter, the maximum in-degree, the number of SCCs, and the number of nodes. We ran all the algorithms using Wolfram Mathematica on a MacBook Pro with an Apple M1 and 8GB RAM. For each type of random network (i.e., Erdős-Rényi, Barabási-Albert, and Watts-Strogatz), we randomly generated 50 networks in the following manner. For five different sets of nodes, we randomly generated ten different networks, where the sets of nodes were 100, 200, 300, 400, and 500 nodes. Furthermore, to generate the random networks, we selected two different sets of parameters for each type of random network.

A. Erdős-Rényi

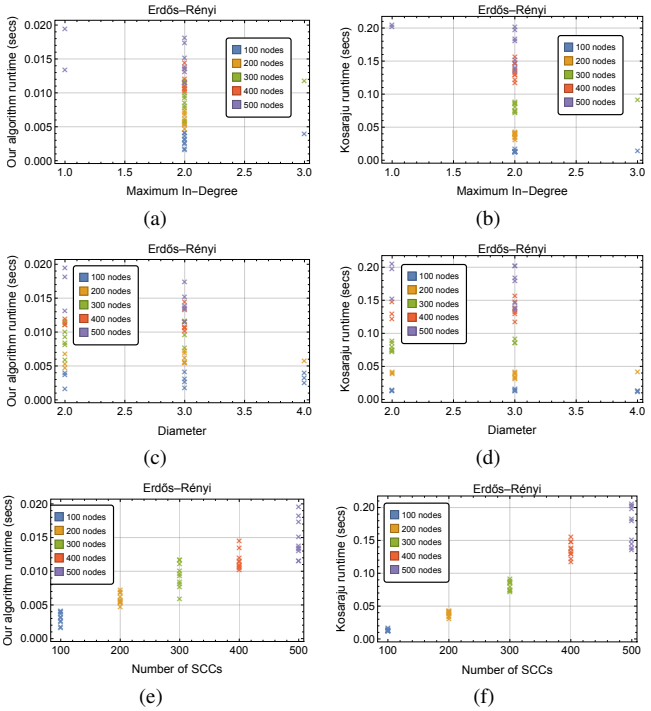

Fig. 4: These figures show the relationship between the network properties of some randomly generated Erdős-Rényi networks and their run times for both our proposed algorithm and Kosaraju's algorithm.

The Erdős-Rényi network requires two parameters, the number of nodes and the number of edges. For the first set of parameters (Figs. 4 and 5), the number of nodes were chosen to be 100, 200, 300, 400, 500, and the number of edges were chosen to be the number of nodes raised to the $2/3$ power. In the second set of parameters (Figs. 6 and 7), again the number of nodes remained the same, but the number of edges was fixed to 500 for all the sets of nodes.

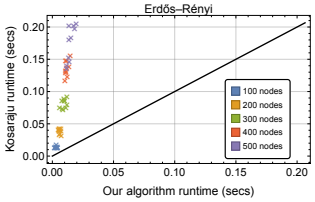

Fig. 5: This figure compares the run times of both our proposed algorithm and Kosaraju's algorithm for several randomly generated Erdős-Rényi networks. We see that our algorithm performs better on networks with a higher number of nodes.

In Fig. 4, we see the comparison between different properties of the network, including the maximum in-degree, diameter, and total number of SCCs, with the run times of both our algorithm and Kosaaraju's algorithm. In this first set of parameters, the number of SCCs plays a much larger role in determining the run-time of the algorithm. In Fig. 5, we see the comparison between the run times of both our algorithm and Kosaraju's algorithm on different randomly generated Erdős-Rényi networks using the first set of parameters. Our algorithm outperforms Kosaraju's.

The results from the second set of parameters for Erdős-Rényi networks are shown in Figs. 6 and 7. In these networks, the diameter and maximum in-degree are much larger, so they increase the runtime of our algorithm. Fig. 7 shows that the runtime of our algorithm only outpeforms the Kosaraju algorithm when there are more nodes in the network.

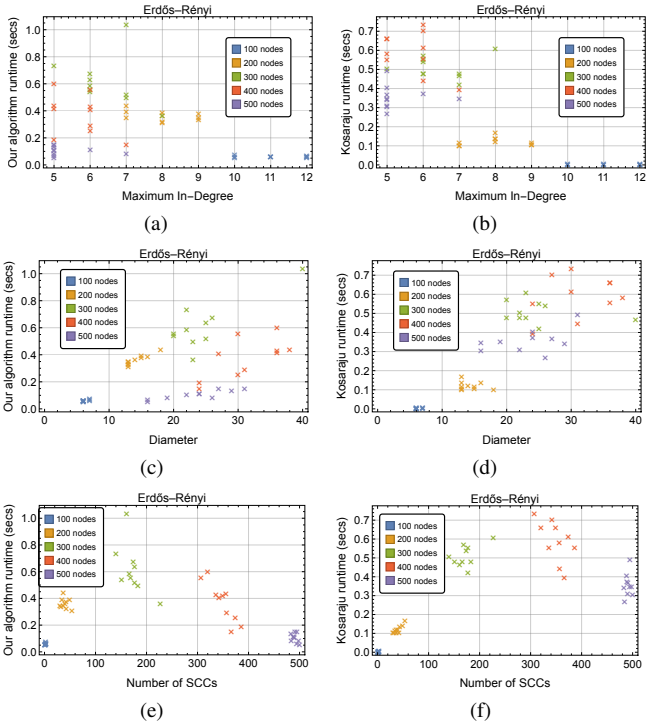

Fig. 6: These figures show the relationship between the network properties of some randomly generated Erdős-Rényi networks and their run times for both our proposed algorithm and Kosaraju's algorithm.

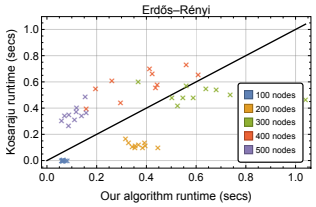

Fig. 7: This figure compares the run times of both our proposed algorithm and Kosaraju's algorithm for several randomly generated Erdős-Rényi networks. We see that our algorithm performs better on networks with a higher number of nodes.

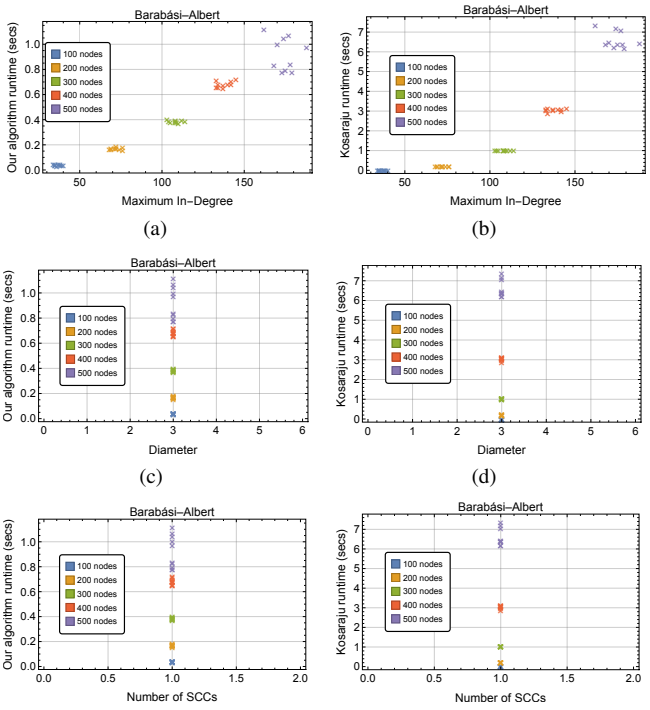

Fig. 8: These figures show the relationship between the network properties of some randomly generated Barabási-Albert networks and their run times for both our proposed algorithm and Kosaraju's algorithm.

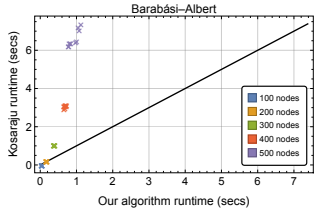

Fig. 9: This figure compares the run times of both our proposed algorithm and Kosaraju's algorithm for several randomly generated Barabási-Albert networks. We see that our algorithm performs better on networks with a higher number of nodes.

B. Barabási-Albert

The Barabási-Albert networks require two parameters, including the number of nodes and the number of edges added to a new vertex at each step.

For the first set of parameters (Figs. 8 and 9), the numbers of nodes were fixed to 100, 200, 300, 400, 500, and the numbers of edges added at each step were chosen to be the numbers of nodes divided by 5. In the second set of parameters (Figs. 10 and 11), again the number of nodes remained the same, but the number of edges added at each step was fixed to 50 for all the sets of nodes.

In Fig. 8, we see that the maximum in-degree plays a larger role in determining the run-time of the algorithm. In Fig. 9, we see the comparison between the run times of our algorithm and Kosaraju's algorithm on different randomly generated Barabási-Albert networks. Our algorithm performs better for networks with more nodes.

The results from the second set of parameters for Barabási–Albert networks are shown in Figs. 10 and 11. The results from the two sets of parameters do not present much difference. Again, our algorithm performs better on networks with a higher number of nodes.

C. Watts-Strogatz

Finally, the Watts-Strogatz networks require the following two parameters, the number of nodes and the rewiring probability.

The first set of parameters included the nodes 100, 200, 300, 400, and 500 with a rewiring probability of 0.8, and the results are shown in Figs. 12 and 13. For the second set of parameters, the set of nodes remained the same, but the rewiring probability was reduced to 0.2 with the results shown in Figs. 14 and 15.

In Fig. 12, we see that the diameter dominates the complexity. In Fig. 13, we see the comparison between the run times of ours and Kosaraju's algorithm on the randomly generated Watts-Strogatz networks. Our algorithm underperforms compared to Kosaraju's.

The results from the second set of parameters for the Watts-Strogatz networks are shown in Figs. 14 and 15. The results from the two sets of parameters do not present much difference. It is clear that Kosaraju's algorithm outperforms our algorithm. We believe that the reason Kosaraju's algorithm outperforms ours has to do with the number of edges in the network.

D. Determining the Diameter of a Network

From the results of Theorem 4, our algorithm can determine the finite digraph diameter of the network. Here, we illustrate the relationship between the number of iterations required before terminating our algorithm compared with the finite digraph diameter plus two.

Fig. 16 shows the results from running our algorithm on the random networks using the second set of parameters. We see that the number of required iterations is identical to the network diameter plus two.

Finally, we compared the runtime of our algorithm with the Floyd-Warshall algorithm [13] on the Erdős-Rényi, Barabási-Albert, and Watts-Strogatz networks. We randomly generated ten different Erdős-Rényi networks, using 25 nodes and 50 edges. For the Barabási-Albert network, we used 25 nodes and 3 edges added to each new

This article has been accepted for publication in IEEE Transactions on Automatic Control. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TAC.2022.3209446

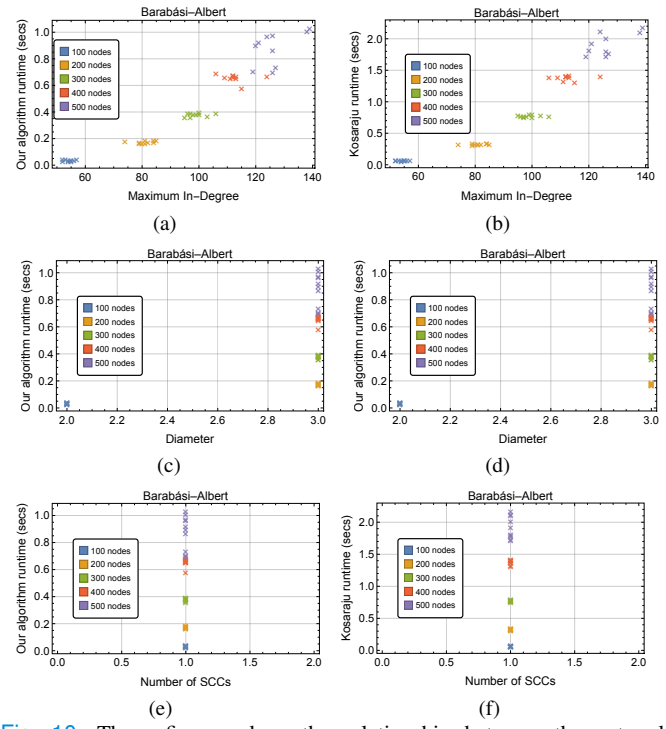

Fig. 10: These figures show the relationship between the network properties of some randomly generated Barabási-Albert networks and their run times for both our proposed algorithm and Kosaraju's algorithm.

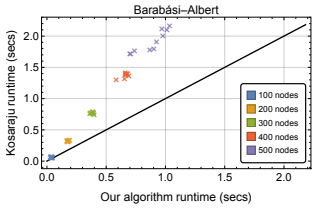

Fig. 11: This figure compares the run times of both our proposed algorithm and Kosaraju's algorithm for different randomly generated Barabási-Albert networks.

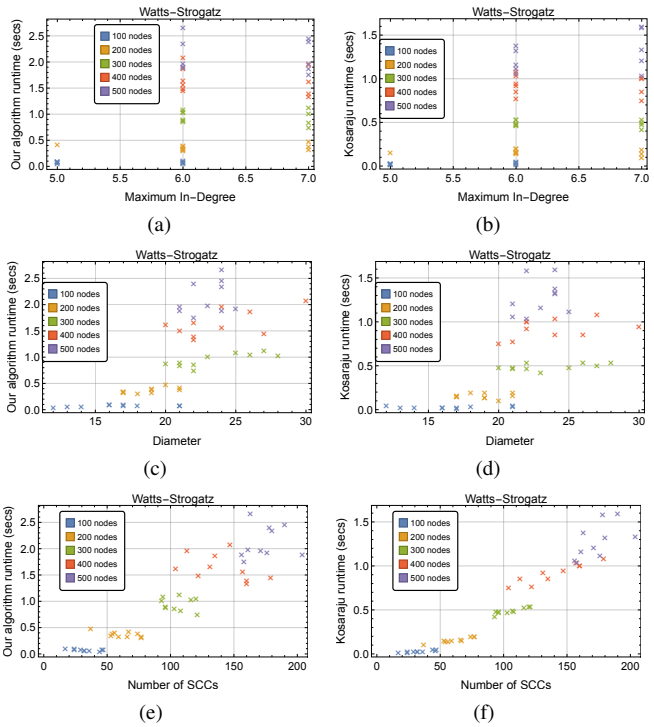

Fig. 12: These figures show the relationship between the network properties of some randomly generated Watts-Strogatz networks and their run times for both our proposed algorithm and Kosaraju's algorithm.

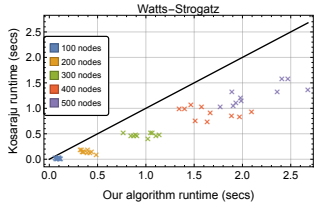

Fig. 13: This figure compares the run times of both our proposed algorithm and Kosaraju's algorithm for several randomly generated Watts-Strogatz networks.

vertex at each time step to generate ten different networks. Finally, we generated ten different Watts-Strogatz networks using 25 nodes and a rewiring probability of 0.2. In Fig. 17, we see that our algorithm outperforms the Floyd-Warshall algorithm for all networks.

E. Discussion

The proposed distributed algorithm to compute the SCCs in a digraph can achieve a lower time complexity than the Kosaraju algorithm in instances when there is a large number of edges that exceed the number of nodes times the finite diameter times the maximum in-degree. In other instances, the Kosaraju algorithm will have a better time complexity. For example, in the worst-case, a network can have its maximum in-degree and finite diameter equal to the number of nodes in the network, which would mean that the time complexity of our distributed algorithm would be $\mathcal{O}(N^3)$. However, the time complexity of the Kosaraju algorithm in the worst case is $\mathcal{O}(N^2)$ since the number of edges could be on the order of N^2. Therefore, we conclude that our distributed algorithm may or may not outperform Kosaraju's depending on the network's topology.

In the case of the centralized algorithm, we provide evidence through exhaustive simulations that suggests that our algorithm outperforms the state-of-the-art Kosaraju algorithm on certain network topologies. However, we remark that this may not always be the case. For instance, when considering the same worst-case network, where the maximum in-degree and finite diameter are equal to the number of vertices in the network, our proposed centralized algorithm will have a time complexity of $\mathcal{O}(N^4)$, which is worse than the time complexity of the Kosaraju algorithm (i.e., $\mathcal{O}(N^2)$ in the worst-case).

When contrasting our proposed distributed algorithm to compute the diameter of a digraph to the state-of-the-art Floyd-Warshall algorithm, we notice that in the worst-case our algorithm's time complexity is no worse than the time complexity of the Floyd-Warshall algorithm, (i.e., $\mathcal{O}(N^3)$). The time complexity for our centralized algorithm is $\mathcal{O}(N^4)$ in the worst case, which is worse than the Floyd-Warshall algorithm in the worst-case; however, our simulations suggest that our centralized algorithm can outperform the Floyd-Warshall algorithm in some instances.

It is important to remark that the results presented here focused on the time-complexity to enable a direct comparison with the algorithms in the literature. Nonetheless, the nature of distributed algorithms requires the assessment of the communication-complexity. Depending on the protocol being used to exchange information (e.g., IDs), it could further increase the complexity by $\mathcal{O}(N\log(k))$, where k is the number of bits needed to transmit the Nth ID. Further investigation should consider the design of suitable communication protocols within their specific applications to improve the overall performance of a new class of distributed algorithms, for which the foundation is laid out in this paper.

VI. Conclusions

We provided for the first time a *scalable distributed* algorithm to find the strongly connected components and finite diameter of a directed network. The proposed solution has a time-complexity $\mathcal{O}\left(ND{d^{\max}_{\text{in-degree}}}\right)$, where N is the number of vertices, D is the finite

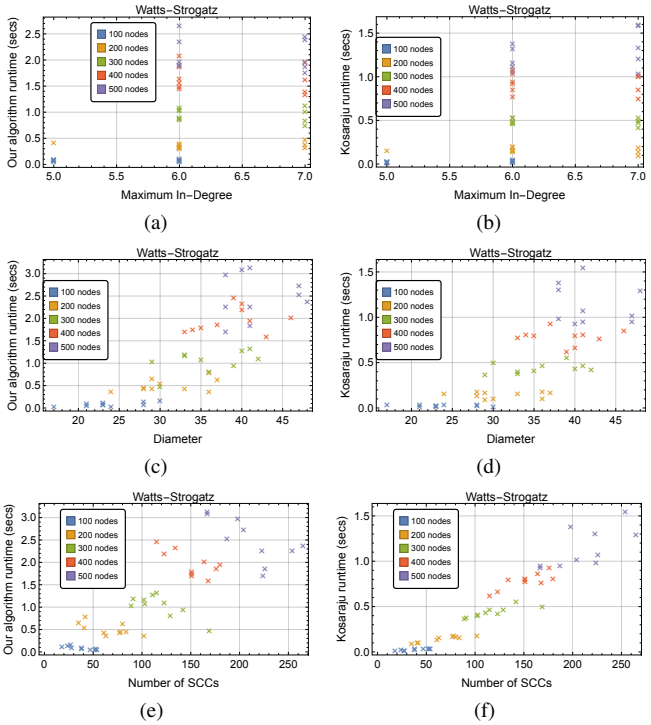

Fig. 14: These figures show the relationship between the network properties of some randomly generated Watts-Strogatz networks and their run times for both our proposed algorithm and Kosaraju's algorithm.

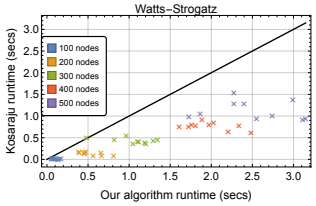

Fig. 15: This figure compares the run times of both our proposed algorithm and Kosaraju's algorithm for several randomly generated Watts-Strogatz networks.

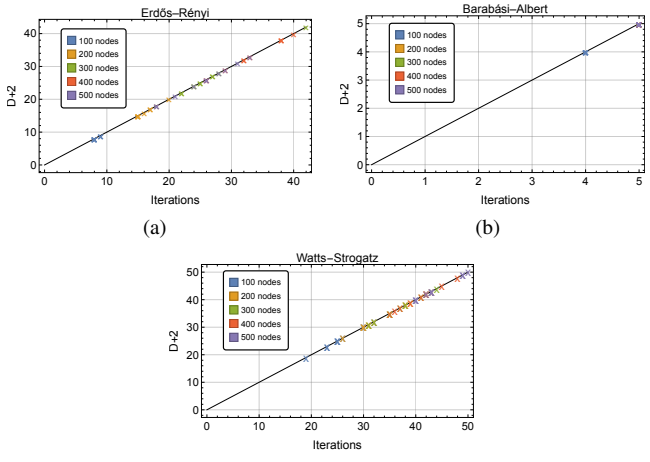

Fig. 16: This figure shows the relationship between the number of iterations needed before terminating our algorithm and the diameter plus two of several randomly generated networks.

diameter and $d^{\max}_{\text{in-degree}}$ is the maximum in-degree of the network. We demonstrated the performance of our centralized algorithm on several random networks. We compared the runtime of our centralized algorithm against Kosaraju's algorithm and found that our centralized algorithm outperformed Kosaraju's for certain network topologies. Additionally, we provided exhaustive simulations that support that our centralized algorithm outperformed Floyd-Warshall's in computing the finite digraph diameter on every tested random network.

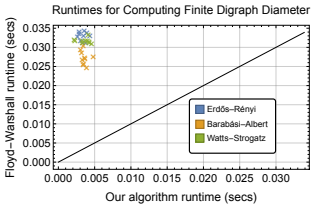

Fig. 17: This figure compares the runtimes of computing the finite digraph diameter when using our algorithm against the Floyd-Warshall algorithm on several randomly generated networks, including Erdős-Rényi, Barabási-Albert, and Watts-Strogatz.

In our future work, we seek to understand how the time complexity may be improved for different types of densely directed networks. Furthermore, we plan to examine how to find the SCCs and diameter while taking privacy into consideration such that the ID of the nodes can be hidden in the process of sharing information. Finally, our future work will focus on developing possible asynchronous protocols capable of determining the strongly connected components and finite diameter of a digraph.

REFERENCES

[1] G. Ramos, A. P. Aguiar, and S. Pequito, "An overview of structural systems theory," *Automatica*, vol. 140, p. 110229, 2022.

[2] F. Bullo, J. Cortés, and S. Martinez, *Distributed Control of Robotic Networks: A Mathematical Approach to Motion Coordination Algorithms*. Princeton University Press, 2009, vol. 27.

[3] R. Albert, H. Jeong, and A.-L. Barabási, "Diameter of the world-wide web," *Nature*, vol. 401, no. 6749, pp. 130–131, 1999.

[4] Y. Xue and P. Bogdan, "Reliable multi-fractal characterization of weighted complex networks: algorithms and implications," *Scientific Reports*, vol. 7, no. 1, pp. 1–22, 2017.

[5] L. Zongxiang, M. Zhongwei, and Z. Shuangxi, "Cascading failure analysis of bulk power system using small-world network model," in *Proceedings of the International Conference on Probabilistic Methods Applied to Power Systems*. IEEE, 2004, pp. 635–640.

[6] J. G. Kuhl and S. M. Reddy, "Distributed fault-tolerance for large multiprocessor systems," in *Proceedings of the 7th Annual Symposium on Computer Architecture*, 1980, pp. 23–30.

[7] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM Journal on Computing*, vol. 1, no. 2, pp. 146–160, 1972.

[8] E. W. Dijkstra, *A Discipline of Programming*. Prentice Hall Englewood Cliffs, 1976, vol. 613924118.

[9] M. Sharir, "A strong-connectivity algorithm and its applications in data flow analysis," *Computers & Mathematics with Applications*, vol. 7, no. 1, pp. 67–72, 1981.

[10] D. F. Hsu, X. Lan, G. Miller, and D. Baird, "A comparative study of algorithm for computing strongly connected components," in *2017 IEEE 15th International Conference on Dependable, Autonomic, and Secure Computing*. IEEE, 2017, pp. 431–437.

[11] L. K. Fleischer, B. Hendrickson, and A. Pinar, "On identifying strongly connected components in parallel," in *International Parallel and Distributed Processing Symposium*. Springer, 2000, pp. 505–511.

[12] J. Barnat, J. Chaloupka, and J. Van De Pol, "Distributed algorithms for strongly connected component decomposition," *Journal of Logic and Computation*, vol. 21, no. 1, pp. 23–44, 2011.

[13] R. W. Floyd, "Algorithm 97: Shortest path," *Communications of the ACM*, vol. 5, no. 6, p. 345, Jun. 1962.

[14] J. Cortés, "Distributed algorithms for reaching consensus on general functions," *Automatica*, vol. 44, no. 3, pp. 726–737, 2008.

[15] H. N. Gabow, "Path-based depth-first search for strong and biconnected components; CU-CS-890-99," *CS Technical Reports. 837.*, 1999.

[16] A. Fronczak, P. Fronczak, and J. A. Hołyst, "Average path length in random networks," *Physics Review E*, vol. 70, p. 056110, Nov 2004.

[17] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world'networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.