Efficiently Learning the Topology and Behavior of a Networked Dynamical System Via Active Queries

Daniel J. Rosenkrantz [1] **Abhijin Adiga** [* 2] **Madhav V. Marathe** [* 3] **Zirou Qiu** [* 3]
S. S. Ravi [* 1] **Richard E. Stearns** [* 1] **Anil Vullikanti** [* 3]

Abstract

Many papers have addressed the problem of learning the behavior (i.e., the local interaction function at each node) of a networked system through active queries, assuming that the network topology is known. We address the problem of inferring both the network topology and the behavior of such a system through active queries. Our results are for systems where the state of each node is from $\{0, 1\}$ and the local functions are Boolean. We present inference algorithms under both batch and adaptive query models for dynamical systems with symmetric local functions. These algorithms show that the structure and behavior of such dynamical systems can be learnt using only a polynomial number of queries. Further, we establish a lower bound on the number of queries needed to learn such dynamical systems. We also present experimental results obtained by running our algorithms on synthetic and real-world networks.

1. Introduction

Many practical problems in social, behavioral and biological sciences can be modeled using networked dynamical systems (also known as graphical dynamical systems) (Laubenbacher & Stigler, 2009; Eubank et al., 2004; Lum et al., 2014; Kauffman et al., 2003; Macy & Willer, 2002; Granovetter, 1978). Such systems also occur naturally in the study of agent-based modeling and analysis (Bonabeau, 2002; Namatame & Chen, 2016), and graphical games (Kearns & Vazirani, 1994; Abouheaf et al., 2014). A networked dynamical system has three components: (i) a set of agents, (ii) a network that specifies how the agents interact and (iii) a local interaction function (behavior) associated with each node. Starting from an initial value (state) for each node, the system evolves in time as each node updates its state based on its own state and the states of its neighbors. The vector consisting of the states of all the nodes at a time t is called the system's **configuration** at time t. The temporal evolution can occur synchronously or asynchronously. For a configuration $\mathcal{C}$ at time t, the configuration $\mathcal{C}'$ at time $t + 1$ is the **successor** of $\mathcal{C}$. Our focus is on networked systems where the system evolves *synchronously* and each local function is Boolean. We refer to these as **synchronous dynamical systems** (SyDSs).

The problem of learning one or more components of a SyDS has been recently studied under various query and observation models. Here, we consider the active query model, where the user queries the unknown system, and based on the responses from the system, attempts to infer its properties. This is similar in form to the teacher model (Goldman & Kearns, 1995; Dasgupta et al., 2019) in the context of concept learning, where a teacher selects instances so that a student can learn the target concept from a concept class. Also, this query model is a more general version of the model used in the literature for learning a Boolean function f (Abasi et al., 2014; Angluin & Slonim, 1994), where each query specifies an input α to f and an oracle returns the value $f(\alpha)$. In all these cases (including ours), the objective is to find a minimum set of queries that is sufficient to learn the object. Several papers have used active querying of networked dynamical systems to learn classes of local functions (e.g., (Adiga et al., 2018; 2020)). In these works, a user provides as input a set **Q** of configurations. The system (or an oracle which has a complete description of the system to be inferred) returns the successor of each configuration in **Q**. The user tries to infer the SyDS from the responses to the queries. Similar work has also been conducted under other query models such as passive observation and probably approximately correct (PAC) model (Adiga et al., 2017; 2019; Narasimhan et al., 2015; Kempe et al., 2003).

[*] Equal Contribution. [1] Biocomplexity Institute and Initiative, University of Virginia, Charlottesville, VA, USA and Department of Computer Science, University at Albany – SUNY, Albany, NY, USA. [2] Biocomplexity Institute and Initiative, University of Virginia, Charlottesville, VA, USA. [3] Biocomplexity Institute and Initiative and Department of Computer Science, University of Virginia, Charlottesville, VA, USA. Correspondence to: S. S. Ravi <ssravi0@gmail.com>, Zirou Qiu <zq5au@virginia.edu>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

The learning frameworks used in the above references assume full knowledge of the network topology. The bounds and algorithms developed in these works rely heavily on the network structure. The focus of our work is on simultaneously learning the network topology and the individual local functions using active queries. This is particularly challenging considering that the only available information to a user is the number of nodes in the network and the model class for the local functions. In this work, our focus is on SyDSs where the local functions are (Boolean) threshold and symmetric functions. These function classes are defined in Section 2.

1.1. Contributions

(1) Learning Threshold- and Symmetric-SyDSs with Arbitrary Topologies: For symmetric-SyDSs (i.e., SyDSs where each local function is symmetric), we present an inference method with no restrictions on the network topology. For any symmetric-SyDS with n nodes, our method uses $O(n^2)$ queries under the batch mode. Under the adaptive mode, we present inference methods for threshold- and symmetric-SyDSs using $O(n + m \log n)$ queries, where m is the number of edges. Thus, if the network is known to be sparse (e.g., $m = O(n)$), the inference algorithm under the adaptive model uses asymptotically fewer queries. The adaptive inference algorithm does not assume that the value of m is known; the parameter m in the upper bound results from the analysis.

(2) A randomized algorithm: For a threshold-SyDS with maximum node degree Δ, we present a randomized algorithm[1] that infers the topology and local functions with high probability using $O(n\Delta \log n)$ queries. This algorithm works in the batch mode. When Δ is a constant (i.e., $m = O(n)$), the number of queries ($O(n \log n)$) used by this algorithm is asymptotically the same as our algorithm for threshold-SyDSs under the adaptive mode. Thus, this algorithm points out that randomization can provide the same benefit as an adaptive algorithm.

(3) A lower bound on the query size under the batch mode: We show that any query set under the batch mode which can correctly infer the topology of any threshold-SyDS with n nodes must be of size $\Omega(n \log n)$. This is done by showing that for any smaller query set $\mathbf{Q}$, one can create two SyDSs $\mathcal{S}$ and $\mathcal{S}'$ with different topologies such that the responses to each query in $\mathbf{Q}$ from the two SyDSs are identical. In other words, such a query set cannot distinguish between the two SyDSs. Since threshold-SyDSs are a subclass of symmetric-SyDSs, this lower bound also applies to Symmetric-SyDSs.

(4) Experimental results: We evaluate the performance of our inference algorithms on threshold-SyDSs using several synthetic as well as real-world networks. In these experiments, we measure the performance of algorithms under the adaptive mode with respect to the number of queries used by the batch query algorithm. We also study how the network density, structure, and threshold assignments influence the number of queries generated by our algorithms.

1.2. Related Work

Learning the components of unknown objects or systems has been an active area of research. For example, many researchers have considered problems related to learning objects such as finite automata (e.g., (Murphy, 1996)), Boolean functions (e.g., (Kearns & Vazirani, 1994; Abasi et al., 2014; Hellerstein & Servedio, 2007; Angluin & Slonim, 1994; Berestovsky & Nakhleh, 2013; Liśkiewicz et al., 2017)) and distributions (e.g., (Kearns et al., 1994)). Active querying to predict users' choices from a known set of options is studied in (Kleinberg et al., 2017). In the context of networked interaction systems, work on inferring behaviors (such as thresholds, polynomial or other influence functions used by nodes) has also been reported (e.g., (González-Bailón et al., 2011; Romero et al., 2011; Laubenbacher & Stigler, 2009; He et al., 2016; Narasimhan et al., 2015)). Abrahao et al. (2013), Gomez-Rodriguez et al. (2010) and Soundarajan & Hopcroft (2010) consider the problem of inferring the network structure given the contagion propagation model.

Algorithms for learning the local functions of discrete dynamical systems from observed data were presented in (Adiga et al., 2017). The (passive) observation model used in that paper does not permit users to actively interact with the system. The active query model for discrete dynamical systems used in our paper has also been used in other papers (e.g., (Adiga et al., 2018; 2019; 2020)). The focus of these papers is on inferring the local functions assuming that the network topology is known. In contrast, our work shows how the queries can be used to learn both the network topology and the local functions. Moreover, the number of queries used by our algorithms is polynomial in the size of the dynamical system.

2. Preliminaries

2.1. Synchronous Dynamical Systems

In presenting the definitions associated with our system model, we follow the notation used in (Adiga et al., 2018). We use $\mathbb{B}$ to denote the Boolean domain {0,1}. A Synchronous Dynamical System (SyDS) $\mathcal{S}$ over $\mathbb{B}$ is a pair $\mathcal{S} = (G, \mathcal{F})$, where (i) $G(V, E)$, an undirected[2] graph with

[1] This algorithm needs an estimate of Δ as one of the inputs. The closer the estimate is to the actual value of Δ, the smaller is the number of queries used by the algorithm.

[2] We will indicate the necessary modifications for directed graphs later in this section.

$|V| = n$, represents the underlying graph of the SyDS, with node set V and edge set E, and (ii) $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$ is a collection of functions in the system, with f_i denoting the **local function** associated with node v_i, $1 \le i \le n$. At any time, each node of G has a state value from $\mathbb{B}$. Each function f_i specifies the interaction between node v_i and its neighbors in G, $1 \le i \le n$. The inputs to function f_i are the state of v_i and those of the neighbors of v_i in G; for each input, the function f_i outputs a value in $\mathbb{B}$, and this value is the next state of v_i. In a SyDS, all nodes compute and update their next state *synchronously*. Other update disciplines (e.g., sequential updates) have also been considered in the literature (Mortveit & Reidys, 2007). At any time t, if $s_i^t \in \mathbb{B}$ is the state of node v_i ($1 \le i \le n$), the **configuration** $\mathcal{C}$ of the SyDS is the n-vector $(s_1^t, s_2^t, \ldots, s_n^t)$. If a SyDS has a one step transition from configuration $\mathcal{C}$ to $\mathcal{C}'$, then $\mathcal{C}'$ is the **successor** of $\mathcal{C}$. Since our local functions are deterministic, the successor of each configuration is *unique*.

Our focus is on two classes of local (Boolean) functions, namely threshold functions and symmetric functions (Crama & Hammer, 2011). For each integer $k \ge 0$, a k-**threshold** function has the value 1 iff at least k inputs are 1. A **symmetric** function depends only on the number of 1's in the input. Thus, a symmetric function with q inputs can be specified by a table T with $q+1$ elements, where $T[i]$ is the value of the function when the number of 1's in the input is i, $0 \le i \le q$. The class of threshold functions is (properly) contained in the class of symmetric functions. Following (Adiga et al., 2018), we use the term **symmetric-SyDS** (**threshold-SyDS**) to denote a SyDS whose local functions are all symmetric (threshold) functions. We now present an example of a threshold-SyDS.

Example: An example of a SyDS with 4 nodes is shown in Figure 1. The threshold value for each node is indicated in blue. The left panel shows the configuration $\mathcal{C}$ at a certain time step, where green and red indicate nodes in state 0 and 1 respectively. The right panel shows the successor of $\mathcal{C}$. For example, the state of node v_3 in the successor is 0 since its threshold is 3, and in $\mathcal{C}$, only two of its inputs (namely, the states of v_1 and v_3) are 1. □

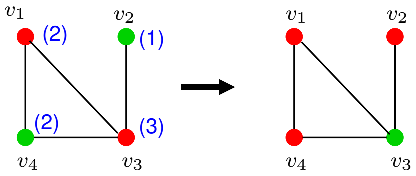

Figure 1. An example of a SyDS where each node has a threshold function. The threshold value for each node is indicated in blue. Green and Red indicate state values of 0 and 1 respectively. The left panel shows the configuration $\mathcal{C}$ at a certain time step and the right panel shows the successor of $\mathcal{C}$.

SyDSs on directed graphs. SyDSs over $\{0, 1\}$ whose underlying graphs are directed are known as **Synchronous Boolean Networks** in the literature (e.g., (Ogihara & Uchizawa, 2017; 2020)), and they have been used to model certain biological phenomena (Kauffman et al., 2003). For any SyDS over a directed graph, the set of inputs to the local function at a node v consists of the state of v and those of its in-neighbors (i.e.. nodes from which v has incoming edges). Except for this, there is no difference between the definitions for SyDSs over undirected and directed graphs.

2.2. Query Model

In general, the input to any of the learning algorithms considered in our work is the set $V = \{v_1, v_2, \ldots, v_n\}$ of the n nodes of the SyDS to be learnt. The algorithm may have additional information regarding the class of local functions and/or restrictions on the network topology. The algorithm must learn both the topology of the underlying network and the local function at each node using suitable queries. The query model used in our work was used in previous work for learning local functions of a SyDS when the topology of the underlying network is known (Adiga et al., 2018; 2019; 2020). Each query Q specifies a configuration of the SyDS $\mathcal{S}$ for which the underlying network and the local functions must be inferred. An oracle (which has a complete description of $\mathcal{S}$) returns the successor Q' of Q. For a SyDS $\mathcal{S}$, a query Q specifies the input to each of the local functions of $\mathcal{S}$, and the successor Q' specifies the value of each local function for the specified input. We consider two well known query modes, namely **batch** and **adaptive** modes (Adiga et al., 2018). In the batch mode, the entire query set $\mathbf{Q}$ must be given to the oracle, and the oracle returns the set $\mathbf{R}$ of responses. In the adaptive mode, queries are specified in multiple stages, and a query at a certain stage may be based on the responses to the queries in previous stages.

2.3. Assumptions, Notation and Terminology

We assume that for any SyDS, the underlying graph is simple (i.e., it has no self loops or multi-edges). We are given the set $V = \{v_1, v_2, \ldots, v_n\}$ of n nodes. However, in general, we don't know either the number or the set E of edges.

For SyDSs on undirected graphs, we use $d(v)$ to denote the degree of node v and d_{avg} to denote the average degree of the graph. The (closed) neighborhood of a node v, denoted by $N[v]$, consists of v and all its neighbors. For SyDSs on directed graphs, we use $N[v]$ to denote the set consisting of v and its in-neighbors and $d(v)$ to denote its in-degree. When considering threshold-SyDSs, we use $\tau(v)$ to denote the threshold of node v. Given any configuration $\mathcal{C}$ and node v, we use $\mathcal{C}[v]$ to denote the state of v in $\mathcal{C}$.

Recall that each query Q is a configuration of the SyDS $\mathcal{S}$ to be inferred. For any node v, we use $Q[v]$ to denote the {0,1} value assigned to v by Q. Note that Q specifies the inputs to each local function of $\mathcal{S}$. Given a query Q and a node v, we use **score**$[Q, v]$ to denote the number of 1's in the input specified by Q for the local function f_v at v. Since our focus is on symmetric functions, we note that score$[Q, v]$ determines the value of f_v for the input specified by Q.

Limitations on learning the topology due to constant functions. A constant-0 (constant-1) Boolean function has the value 0 (1) for all inputs. When a SyDS on an undirected graph has two nodes u and v such that both the local functions f_u and f_v are constant functions, the query model cannot be used to identify whether or not the edge $\{u, v\}$ is in the underlying graph. To see why, let $\mathcal{S}$ and $\mathcal{S}'$ denote two SyDSs which are identical except that $\mathcal{S}$ has the edge $\{u, v\}$ while $\mathcal{S}'$ does not have that edge. It can be verified that for any query Q, the responses for $\mathcal{S}$ and $\mathcal{S}'$ are identical. So, learning algorithms under the model where each query Q is a configuration and the response is the successor of Q, cannot distinguish between $\mathcal{S}$ and $\mathcal{S}'$. Likewise, for SyDSs on directed graphs, if the local function at a node v is a constant function, then none of the incoming edges to v can be identified using the query model.

A note about constant functions for threshold-SyDSs. For any node v with degree $d(v)$, the number of inputs to the local function f_v is $d(v) + 1$. For a threshold-SyDS, if the local function at a node v is the constant-0 function, its threshold is assumed to be $d(v)$+2. Likewise, if the local function at v is the constant-1 function, the threshold of v is assumed to be 0. Thus, the threshold $\tau(v)$ of any node v satisfies the condition $0 \leq \tau(v) \leq d(v) + 2$.

3. Anchor Nodes and Their Role in Inference

Several of our algorithms for learning the topology rely on identifying for each node v_i, an anchor node v_j. We will first explain this idea in the context of threshold-SyDSs. We also describe a simple inference algorithm called ANCHORLINEARSEARCH that makes use of this idea. The idea of using anchor nodes and sequential search to identify the neighbors of nodes can also be used for symmetric-SyDSs, as will be shown in Section 4.1. For simplicity, we assume in this section that there no nodes with constant functions.

Consider a given ordering $\langle v_1, v_2, \ldots, v_n \rangle$ of the nodes and the following set $\mathbf{Q} = \{Q_1^1, Q_2^1, \ldots, Q_n^1\}$ of n queries, where Q_j^1 sets nodes v_1 through v_j to 1 and the remaining nodes (if any) to 0, $1 \leq j \leq n$. Let $\mathbf{R} = \{R_1^1, R_2^1, \ldots, R_n^1\}$ denote the responses produced for the query set $\mathbf{Q}$. For a node v_i with threshold $\tau(v_i)$, suppose j is the *smallest* index such that $R_j^1[v_i] = 1$; thus, the state of v_i is 1 for the first time (in the chosen ordering) when the response to Q_j^1 is obtained. We refer to v_j as the **anchor** node for v_i. Given the definition of threshold, we have the following observation regarding the anchor node.

Observation 3.1. Consider a threshold-SyDS $\mathcal{S}$ with no constant local functions. Given the ordering $\langle v_1, v_2, \ldots, v_n \rangle$ of the nodes of $\mathcal{S}$, let v_j be the anchor node for v_i. The subset $\{v_1, v_2, \ldots, v_j\}$ includes exactly $\tau(v_i)$ nodes in the closed neighborhood $N[v_i]$ of v_i. Further, $v_j \in N[v_i]$. ■

If $v_i \neq v_j$, we conclude that the edge $\{v_i, v_j\}$ is in the underlying graph. Given the anchor node v_j for v_i, a simple method to identify the other neighbors of v_i is as follows.

(1) First, consider the nodes v_1, v_2, …, v_{j-1} to the left of v_j. For each k, $1 \leq k \leq j - 1$, consider the query Q_{ik} which sets all the nodes in $\{v_1, v_2, \ldots, v_j\} - \{v_k\}$ to 1 and all the other nodes to 0. Let R_{ik} be the response to Q_{ik}. Using Observation 3.1, it can be verified that $v_k \in N[v_i]$ iff $R_{ik}[v_i] = 0$. Thus, using $j - 1$ queries of the form Q_{ik}, the set $N_L[v_i]$ of all the neighbors of v_i in $\{v_1, v_2, \ldots, v_j\}$ can be found. Also, $|N_L[v_i]|$ gives the threshold of v_i.

(2) Now, consider the nodes v_{j+1}, v_{j+2}, …, v_n to the right of v_j. To find the neighbors of v_i among these nodes, we do the following. For each k, $j + 1 \leq k \leq n$, we use the query Q_{ik}, in which each node in $(N_L[v_i] - \{v_j\}) \cup \{v_k\}$ is set to 1, and all the other nodes are set to 0. it can be verified that $v_k \in N[v_i]$ iff $R_{ik}[v_i] = 1$. Thus, using $n - j$ queries, the set $N_R[v_i]$ of all the neighbors of v_i in $\{v_{j+1}, v_{j+2}, \ldots, v_n\}$ can be found.

Thus, once we have an anchor for a node v_i, all the neighbors of v_i and $\tau(v_i)$ can be found. The above method uses $O(n)$ queries for each node v_i using a simple sequential search based on the anchor for v_i.

Inference algorithm ANCHORLINEARSEARCH. It can also be seen from the above discussion that if all the nodes in a subset W have the *same* anchor node v_j, the n queries used in the sequential search method can be used to find all the neighbors of each node in W. Hence, if the number of distinct anchor nodes identified by the query set $\mathbf{Q}$ is k, then all the edges of the underlying graph and the thresholds of all the nodes can be found using $O(kn)$ queries. In our experiments, we refer to this as ANCHORLINEARSEARCH. Instead of sequential search, one can use binary search. The corresponding algorithm ANCHORBINARYSEARCH is described in Section 4.2.

A lower bound on the number of anchor nodes: Suppose $\mathcal{S}$ is a SyDS whose underlying undirected graph $G(V, E)$ has n nodes and maximum node degree Δ. From the definition of an anchor node, it is seen that each node v can be the anchor for only the nodes in its closed neighborhood $N[v]$. Since the maximum node degree is Δ, for each $v \in V$, $|N[v]| \leq \Delta + 1$. Thus, we have:

Observation 3.2. For any SyDS $\mathcal{S}$ on an undirected graph $G(V, E)$ with n nodes and maximum node degree Δ, the number of anchor nodes needed for G is $\geq \lceil n/(\Delta + 1) \rceil$. ■

4. Deterministic Learning Algorithms

We present two learning algorithms in this section. The first algorithm is for learning symmetric-SyDSs under the batch mode while the second algorithm is for learning threshold-SyDSs under the adaptive mode. A generalization of the adaptive mode algorithm to symmetric-SyDSs is given in Section A.2 of the supplement.

4.1. Learning Symmetric-SyDSs Under the Batch Mode

Here, we show that the topology and local functions of any symmetric-SyDS on n nodes can be learnt under the batch mode using $O(n^2)$ queries. We first present our algorithm for directed graphs and then indicate the necessary modifications for undirected graphs. The pseudocode for our learning algorithm, called Alg-Dir-Sym-Batch, appears in Figure 2. Step 1 constructs the query set $\mathbf{Q}$, and it can be seen that $|\mathbf{Q}| = O(n^2)$. Step 3(a) identifies the anchor node for each node v_p using the response to the query set[3] $\mathbf{Q}^1$ (as discussed in Section 3). Steps 3(b), (c) and (d) identify the incoming edges to each node v_p using the responses to the queries in $\mathbf{Q}^2 \cup \mathbf{Q}^3$. Thus, Step 4 can compute the indegree and the (closed) in-neighborhood of each node v_p. Finally, Step 5 computes the symmetric function for each node using the responses for the queries in $\mathbf{Q}^1$. The following theorem (whose proof appears in Section A.1 of the supplement) establishes the correctness of the algorithm.

Theorem 4.1. *Let $\mathcal{S}$ be a symmetric-SyDS on a* directed *graph with node set $V = \{v_1 \ldots, v_n\}$. Then in polynomial time, a batch query set Q with $O(n^2)$ queries can be constructed, such that Q can be used to identify the local functions of $\mathcal{S}$, and for each node v whose local function is not a constant function, the set of edges entering v.*

Modifications for undirected graphs. If the graph of $\mathcal{S}$ is undirected, then the identification of $\mathcal{S}$ can be simplified as follows. The query set $\mathbf{Q}$ is still used, and the responses to the queries in $\mathbf{Q}^1$ are processed in the same way. We then process the nodes whose local function is not a constant function, in increasing order of their indices. Consider the processing of a given node v_p such that f_p is not a constant function. At this point, we have already processed all nodes v_j where $j < p$ and f_j is not a constant function. So, if $j < p$ and f_j is not a constant function, whether or not there is an edge between v_j and v_p has already been determined. Let v_r be the anchor node for v_p. If $j < p$ and f_j is a constant function, we use the value of $R^2_{r,j}[v_p]$ to determine whether of not there is an edge between v_j and v_p. If $j > p$, we use the value of $R^3_{r,j}[v_p]$ to determine whether of not there is an edge between v_j and v_p.

```
1. Construct the following query sets $\mathbf{Q}^1$, $\mathbf{Q}^2$ and $\mathbf{Q}^3$:
   (a) $\mathbf{Q}^1$ has $n+1$ queries denoted by $Q^1_i$, $0 \le i \le n$.
       In query $Q^1_0$, all the $n$ bits are 0. For $1 \le i \le n$, in
       query $Q^1_i$, the first $i$ bits are 1 and the rest (if any) are 0.
   (b) $\mathbf{Q}^2$ has $n(n-1)/2$ queries denoted by $Q^2_{i,j}$, where
       $2 \le i \le n$ and $1 \le j \le i-1$. In query $Q^2_{i,j}$, the
       first $i$ bits, except for bit $j$, are 1; bit $j$ and all the
       remaining bits are 0.
   (c) $\mathbf{Q}^3$ has $n(n-1)/2$ queries denoted by $Q^3_{i,j}$, where
       $1 \le i \le n-1$ and $i+1 \le j \le n$. In query $Q^3_{i,j}$, the
       first $i-1$ bits and the bit $j$ are 1; all the remaining bits are 0.
2. Submit the query set $\mathbf{Q} = \mathbf{Q}^1 \cup \mathbf{Q}^2 \cup \mathbf{Q}^3$ to the system and obtain
   the response set $\mathbf{R} = \mathbf{R}^1 \cup \mathbf{R}^2 \cup \mathbf{R}^3$. Initialize $E = \emptyset$.
3. for $p = 1$ to $n$ do
   (a) Let $r$ = Min $\{k \ge 1 : R^1_0[v_p] \ne R^1_k[v_p]\}$. (Note:
       $v_r$ is the anchor node for $v_p$.)
   (b) if $(p \ne r)$ then $E = E \cup \{(v_r, v_p)\}$.
   (c) for $j = 1$ to $r-1$ do
       (Note: Finds all the incoming edges to $v_p$ from
       $\{v_1, \ldots, v_{r-1}\}$.)
           if $((p \ne j)$ and $(R^2_{r,j}[v_p] = R^1_0[v_p]))$
             then $E = E \cup \{(v_j, v_p)\}$.
   (d) for $j = r+1$ to $n$ do
       (Note: Finds all the incoming edges to $v_p$ from
       $\{v_{r+1}, \ldots, v_n\}$.)
           if $((p \ne j)$ and $(R^3_{r,j}[v_p] \ne R^1_0[v_p]))$
             then $E = E \cup \{(v_j, v_p)\}$.
4. Find the indegree $d(v_p)$ and the closed in-neighborhood
   $N[v_p]$ of each node $v_p$ in $G(V, E)$.
5. for $p = 1$ to $n$ do (Note: Constructs the symmetric
   function table $T_p$ for each node $v_p$.)
   (a) for $j = 0$ to $d(v_p) + 1$ do
         (i) Let $k$ be the smallest integer such that in $Q^1_k$,
             exactly $j$ nodes from $N_p$ are set to 1.
         (ii) $T_p[j] = R^1_k[v_p]$.
```

Figure 2. Description of Algorithm Alg-Dir-Sym-Batch for learning Symmetric-SyDSs on directed graphs

4.2. Adaptive Inference for Threshold-SyDSs

In Section 3, we presented AnchorLinearSearch. The batch mode algorithm in Section 4.1 for symmetric-SyDSs essentially implements that sequential search strategy. We now discuss how one can derive an inference algorithm under the adaptive mode by replacing sequential search by binary search to find the neighbors of a node. For sparse graphs, this adaptive strategy uses asymptotically fewer

[3] The query set $\mathbf{Q}^1$ includes the query in which every node has the value 0 since we need to find the value of each symmetric function when the number of 1's in the input is zero.

queries. We refer to this as ANCHORBINARYSEARCH and present its details for undirected graphs.

Consider a threshold-SyDS, where the ordered node set is $\langle v_1, v_2, \ldots, v_n \rangle$. We first use the query set $\mathbf{Q}^1$ (defined in Figure 2) and get the response set $\mathbf{R}^1$. Note that query $Q_1^0 \in \mathbf{Q}^1$ sets all the nodes to 0 and that query $Q_1^n \in \mathbf{Q}^1$ sets all the nodes to 1. For any node v for which $R_1^0[v] = 1$, the local function f_v is the constant-1 function (i.e., $\tau(v) = 0$). Likewise, for any node v for which $R_1^n[v] = 0$, the local function f_v is the constant-0 function (i.e., $\tau(v) = n + 1$). Thus, we can identify all the nodes with constant functions. All the remaining nodes have non-constant local functions.

As in Section 3, query set $\mathbf{Q}^1$ will also find the anchor for each node. Let v_j be the anchor for v_i. Thus, from Observation 3.1, the subset $\{v_1, \ldots, v_j\}$ contains exactly $\tau(v_i)$ nodes of $N[v_i]$ and $v_j \in N[v_i]$. We want to find the members of $N[v_i]$ in $V_j^L = \{v_1, \ldots, v_{j-1}\}$ (i.e., nodes to the left of v_j) and those in $V_j^R = \{v_{j+1}, \ldots, v_n\}$ (which are to the right of v_j). We will discuss the binary search method for V_j^L; a similar method can be used for V_j^R.

Initially, the search range is $\langle v_1, \ldots, v_{j-1} \rangle$. We will find the leftmost node (i.e., the smallest a such that $1 \le a \le j - 1$ and node v_a in $N[v_i]$) in this range. To do this, let $b = \lceil j/2 \rceil$ (i.e., a midpoint of the current range). Construct the query Q which sets the nodes in $\{v_{b+1}, \ldots, v_j\}$ to 1 and all the other nodes to 0. (In particular, in this query set all the nodes in $\{v_1, \ldots, v_b\}$ to 0.) In the response R to this query, it can be seen that the range $\{v_1, \ldots, v_b\}$ contains a member of $N[v_i]$ iff $R[v_i] = 0$. Therefore, depending on the response to Q, the search range for the next query is either $\langle v_1, \ldots v_b \rangle$ or $\langle v_{b+1}, \ldots v_{j-1} \rangle$. This reduces the size of the search range by a factor of 2. Applying this method recursively, it follows that with $O(\log n)$ queries, we can find the leftmost element of $N[v_i]$ in V_j^L. By repeating this process, we can find each member of $N[v_i] \cap V_j^L$. A similar process can be used to identify each member of $N[v_i] \cap V_j^R$.

We can analyze the algorithm in two ways yielding two different bounds on the number of queries. The first bound of $O(n + m \log n)$ is as follows. Note that for node v_i, $O(\log n)$ queries are used to find each node in $N[v_i]$. Thus, the total number of queries for each node v_i = $O(d(v_i) \log n)$. Therefore, the total number of queries used for binary search over all the nodes is $O(\sum_{v_i \in V} d(v_i) \log n)$ = $O(m \log n)$, since the sum of the node degrees is $2m$, where m is number of edges in the graph. With the initial set $\mathbf{Q}^1$ of $n + 1$ queries used to find the anchor nodes, the total number of queries used in this adaptive inference method is $O(n + m \log n)$. When $m = O(n)$, the number of queries used by this algorithm is asymptotically less than that used by ANCHORLINEARSEARCH. However, when $m = \Theta(n^2)$, it can do worse than ALG-DIR-SYM-BATCH by a factor of $\log n$. The second bound is $O(kn \log n)$, which follows by noting that for each anchor node at most n binary searches are performed. The following theorem summarizes the above discussion.

Theorem 4.2. *Let $\mathcal{S}$ be a threshold-SyDS. Algorithm* ANCHORBINARYSEARCH *infers the threshold values of all the nodes, and each edge $\{u, v\}$ where at least one of f_u and f_v is a non-constant local function. The following are the upper bounds for the number of queries used by the algorithm: (i) $O(n + m \log n)$ and (ii) $O(kn \log n)$.*

The above binary search method for threshold-SyDSs can also be extended to symmetric-SyDSs. The extended algorithm appears in Section A.2 of the supplement.

5. A Randomized Algorithm

We briefly describe how the idea of anchor nodes can be used to design a randomized algorithm to solve the inference problem for threshold-SyDSs under the batch mode. This improves on the deterministic $\Theta(n^2)$ bound (Section 4.1).

We assume that the maximum degree Δ (or an upper bound on it) is known. The algorithm involves the following steps: we run the first step of the algorithm for finding anchor nodes in Section 3 using multiple random permutations of nodes. Specifically, for a permutation $\pi_\ell = \langle v_{\pi_\ell(1)}, v_{\pi_\ell(2)}, \ldots, v_{\pi_\ell(n)} \rangle$, for $\ell = 1$ to $N = 3(\Delta + 1) \log n$, we run the n queries given by $\mathbf{Q}^\ell = \{Q_1^\ell, Q_2^\ell, \ldots, Q_n^\ell\}$, where Q_j^ℓ sets nodes $v_{\pi_\ell(1)}$ through $v_{\pi_\ell(j)}$ to 1 and the remaining nodes (if any) to 0, $1 \le j \le n$. For each permutation ℓ and each node v_i, we identify an anchor node v_j for the query set $\mathbf{Q}^\ell$; if $v_j \ne v_i$, we infer the edge $\{v_i, v_j\}$. Let G denote the graph inferred after all the N permutations. Finally, we run the algorithm of (Adiga et al., 2018) to infer the node threshold values in G; this step uses n queries.

Theorem 5.1. *The above algorithm correctly infers the graph and all threshold functions in a SyDS, with probability at least $1 - 1/n$, using $O(n\Delta \log n)$ batch queries.*

Proof: The number of queries stated in the theorem follows immediately. The algorithm can make a mistake in inferring the graph, if there exists a node v_i with a neighbor v_j, which never becomes an anchor for v_i in any permutation. We show below that this probability is very small.

Observe that a node v_j will be an anchor node in a permutation π_ℓ if it is in position $\tau(v_i)$ (relative to the other neighbors of v_i). The probability of this happening is $1/(d(v) + 1)$. Therefore, the probability that node v_j will not be an anchor in any of the N permutations is $\left(1 - \frac{1}{d(v)+1}\right)^N \le \left(1 - \frac{1}{\Delta+1}\right)^N \le e^{-3 \log n} = \frac{1}{n^3}$, since $N = 3(\Delta + 1) \log n$. By a union bound, the probability that there exists a node v_i for which a neighbor v_j does not become an anchor in any permutation is at most $n^2/n^3 = 1/n$, and the theorem follows. ■

6. A Lower Bound for Batch Mode

In this section, we establish a lower bound on the number of queries needed to learn threshold-SyDSs. Since the class of threshold functions is a subclass of symmetric functions, this lower bound also applies to symmetric-SyDSs.

Theorem 6.1. *Under the batch mode, any query set that can correctly learn the network topology of any threshold-SyDS with n nodes must contain $\Omega(n \log n)$ queries.*

Proof: Let $\mathbf{Q}$ be a query set for threshold SyDSs over a node set V with n nodes. Let $\{\mathbf{Q}^0, \mathbf{Q}^1, \mathbf{Q}^2, \ldots, \mathbf{Q}^n\}$ be a partition of $\mathbf{Q}$, where $\mathbf{Q}^i$ denotes the set of queries in which exactly i nodes are in state 1. We will show that for $\mathbf{Q}$ to correctly infer the network topology of any threshold-SyDS with n nodes, $\mathbf{Q}^t$ must contain at least n/t queries, for each t, $1 \le t \le n$. This will allow us to conclude that $|\mathbf{Q}| = \Omega(n \log n)$.

For the sake of contradiction, suppose $|\mathbf{Q}^t| < n/t$ for some t, $1 \le t \le n$. Let $V_t = \{v \in V \mid \exists Q \in \mathbf{Q}^t$ such that $Q[v] = 1\}$; that is, a node v is in V_t only if there is at least one query $Q \in \mathbf{Q}^t$ such that $Q[v] = 1$. Since $|\mathbf{Q}^t| < n/t$ and each query in $\mathbf{Q}^t$ contributes at most t nodes to V_t, we have $|V_t| < n$. Hence, there is at least one node v_* for which $Q[v_*] = 0, \forall Q \in \mathbf{Q}^t$. Let v_c be any node other than v_*. Let $\mathcal{S}$ be a threshold-SyDS where $G(V, E)$ is a star graph with v_c as the center node that is adjacent to all other nodes. Let $\mathcal{S}'$ be another threshold SyDS where the underlying graph G' is the same as G except that v_c is not adjacent to v_*. In both SyDSs, the threshold of v_c is t while the thresholds of the other nodes are 0. We now show that $\mathcal{S}$ and $\mathcal{S}'$ produce the same response to the queries in $\mathbf{Q}$ and therefore, cannot be distinguished from one another.

Let $\mathbf{Q}^{<t} = \bigcup_{i<t} \mathbf{Q}^i$ denote the set of queries in $\mathbf{Q}$ in which at most $t-1$ nodes are state 1. Similarly, let $\mathbf{Q}^{>t} = \bigcup_{i>t} \mathbf{Q}^i$. For a query Q, let R denote its successor. Also, let $Q[v]$ and $R[v]$ denote the states of node v in Q and R respectively. Note that for any query Q, the response of any $v \in V \setminus \{v_c\}$ is 1; i.e., $R[v] = 1$ in both $\mathcal{S}$ and $\mathcal{S}'$ as its threshold is 0 in both SyDSs. Therefore, we focus on v_c.

First, consider $\mathcal{S}$. Let $d(v)$ and $N[v]$ denote the degree and the closed neighborhood of node v respectively. Let $Q \in \mathbf{Q}^{<t}$. Since $\tau(v_c) = t$ and Q has at most $t-1$ nodes of B in state 1, $R[v_c] = 0$. Now let $Q \in \mathbf{Q}^t \cup \mathbf{Q}^{>t}$. Since $\tau(v_c) = t$, $N[v_c] = V$, and at least t nodes are in state 1, it follows that $R[v_c] = 1$.

Now consider $\mathcal{S}'$. For $Q \in \mathbf{Q}^{<t}$, using the same argument as before, it follows that $R[v_c] = 0$. Let $Q \in \mathbf{Q}^{>t}$. Since $\tau(v_c) = t$, $N[v_c] = V \setminus \{v_*\}$, and at least $t+1$ nodes are in state 1, it follows that at least t nodes from $N[v_c]$ are in state 1. Therefore, $R[v_c] = 1$. Finally, let $Q \in \mathbf{Q}^t$. We recall that $Q[v_*] = 0$. Therefore, all the t nodes that are in state 1 belong to $V \setminus \{v_*\}$, which is same as $N[v_c]$. Therefore, $R[v_c] = 1$.

From the above arguments, it follows that the responses of $\mathcal{S}$ and $\mathcal{S}'$ are identical for each query in $\mathbf{Q}$. In other words, the query set $\mathbf{Q}$ cannot correctly infer whether the edge $\{v_c, v_*\}$ is in the underlying graph of the SyDS. This contradicts the assumption that $\mathbf{Q}$ correctly infers the topology of any threshold-SyDS whose underlying graph has n nodes. Therefore, $|\mathbf{Q}^t| \ge n/t$ for any t, $1 \le t \le n$.

Now, $|\mathbf{Q}| = \sum_{t=0}^{n} |\mathbf{Q}^t| \ge \sum_{t=1}^{n} n/t$. It is well known that the Harmonic sum $\sum_{t=1}^{n} 1/t = \Omega(\ln n)$ (Cormen et al., 2009). Hence, $|\mathbf{Q}| = \Omega(n \log n)$. ■

7. Experimental Results

In this section, we investigate the performance of the proposed inference algorithms for threshold-SyDSs with respect to the structure of the underlying network and threshold assignments. The main algorithms used for experimentation are ANCHORLINEARSEARCH and ANCHORBINARYSEARCH. Since the number of queries required by ALG-DIR-SYM-BATCH is fixed for a given network, we use it as a reference to evaluate other algorithms. When comparing networks with same number of nodes we use **query ratio** $\gamma = a/b$ to evaluate a target adaptive inference algorithm $\mathcal{A}$, where a and b are the numbers of queries used by $\mathcal{A}$ and ALG-DIR-SYM-BATCH respectively. In all experiments, we use the method of Adiga et al. (2018) to assign node thresholds. Given a parameter $\theta \in [0, 1]$, each node v is assigned an integer threshold uniformly at random from the interval $[(d(v)+2)(1-\theta)/2, (d(v)+2)(1+\theta)/2]$. For $\theta = 0$, the threshold is $\lfloor (d(v)+2)/2 \rfloor$ (majority rule) while for $\theta = 1$, a threshold is picked uniformly at random from the interval $[0, d(v)+2]$. The parameter θ controls the deviation of the threshold function from the majority rule; larger the θ, the greater the range of values from which the threshold is picked around $(d(v)+2)/2$.

Datasets, computing and reproducibility. We selected the set of real world networks (shown in Table 1) based on their domains and sizes. Further, we evaluated the proposed algorithms on two classes of synthetic networks: (i) Erdős-Rényi (ER) networks and (ii) Barabási-Albert (BA) networks. Experiments were performed on Intel Xeon(R) machines with 64GB of RAM. The source code and selected datasets can be found at `https://github.com/bridgelessqiu/Inference-ICML`.

Experiments. We performed four sets of experiments. In all these cases, we considered three values of threshold parameter θ, namely 0, 0.5, and 0.9. First, we experimented

Table 1. List of networks

Dataset	Type	n	m	*Max deg*	*Average deg*
`lastfm`	Social	7,624	27,806	216	7.29
`gnutella`	Peer-to-Peer	10,876	39,994	103	7.35
`astroph`	Coauthor	17,903	196,972	504	22
`facebook`	Social	22,470	170,823	709	15.2
`Deezer`	Social	28,281	92,752	172	6.55

with Erdős-Rényi graphs. The number of nodes n was fixed at 1000 and the edge probability p was increased. The second set of experiments correspond to random power-law networks generated using the Barabási–Albert model. Specifically, we studied how a network property such as degree distribution affects k, the number of anchor nodes. We varied the average degree by varying the number of edges that can be incident on the new node added to the network in each iteration. The third set of experiments is on real-world networks. In all these experiments, for each network we chose a random order of nodes and used it for every θ value for fair comparison. Finally, we performed experiments on how the number of anchor nodes varies for random orderings of the nodes of chosen networks.

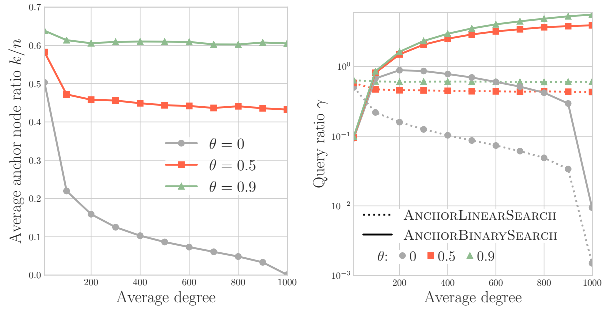

Figure 3. Results for ER graphs: The results are for $n = 1000$, with the average degree varied from 10 to 999 by increasing the edge probability p. For each instance, we used 10 replicates. The average number of anchor nodes and the query ratio are plotted with respect to average degree in the first and second plots respectively. The maximum standard deviations of query ratio for AnchorLinearSearch and AnchorBinarySearch are 0.0132 and 0.0998, respectively. The maximum standard deviation of anchor node ratio is 0.013.

Number of anchor nodes, network density, and thresholds. Recall that the number of anchor nodes k influences the number of queries used by both AnchorLinearSearch and AnchorBinarySearch. Here, we study how it varies with the number of edges in the network and threshold assignments through two studies. The results for Erdős-Rényi graphs in Figure 3 (left panel) show the influence of average degree and threshold parameter θ. We note that for $\theta = 0$, k decreases rapidly with increasing average degree. This decrease can be partially explained by Observation 3.2; as the edge probability increases, the average node degree also increases. For higher θ, even though k decreases with average degree, the rate of decrease is much smaller. We can draw insight from the following extreme example. Consider the complete graph on n nodes, where each node is assigned a distinct threshold. If the nodes are in increasing order of threshold, then $k = n$. Therefore, greater the variation in threshold, the larger is the value of k. Similar observations for BA networks are shown in Figure 4.

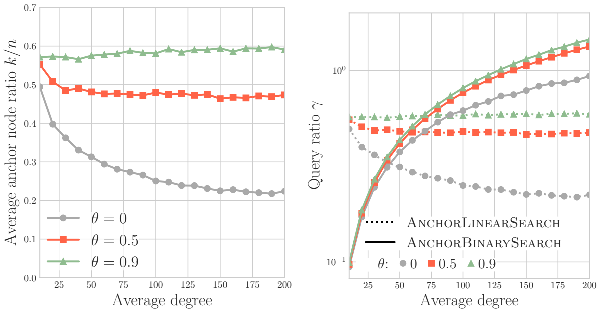

Figure 4. Results for BA networks: The results are for $n = 1000$, with the average degree varied from 10 to 200. For each instance, we used 10 replicates. The average number of anchor nodes and the query ratio are plotted with respect to average degree in the first and second plots respectively. The maximum standard deviations of query ratio for AnchorLinearSearch and AnchorBinarySearch are 0.016 and 0.023, respectively. The maximum standard deviation for anchor node ratio is 0.013.

Comparison between AnchorLinearSearch and AnchorBinarySearch. The number of queries in AnchorLinearSearch directly depends on k (number of queries is $O(kn)$). We observe in Figure 3 (right panel) that the query ratio is strongly correlated with k. Also, since $k \leq n$, it follows that the query ratio will always be less than 1. In contrast, the number of queries in the case of AnchorBinarySearch depends not only on k but also the number of edges m (see Theorem 4.2). Hence, there is a regime (average degree $\leq$ 50) where the number of queries required is much less when compared to AnchorLinearSearch. However, as m increases, the number of queries increases quickly due to the additional factor of $\log n$ contributed by the binary search (first bound in Theorem 4.2). Therefore, we observe that for very high average degree, the number of queries required can exceed that for Alg-Dir-Sym-Batch. (When m is comparable to n^2, the number of queries will be of the order of $n^2 \log n$.) However, we note for $\theta = 0$, that there is a point where the number of queries starts decreasing. This is because of the fact that k is decreasing (second bound in Theorem 4.2).

Querying real-world networks. Since all the real-world networks considered here (Table 1) have low average degree

(< 25), ANCHORBINARYSEARCH outperforms ANCHORLINEARSEARCH. Hence, in Figure 5, we present only the results for ANCHORBINARYSEARCH. We note that the anchor node ratio increases with average degree generally, but is more pronounced for lower θ (left panel). This is in line with the experiments with synthetic networks. For the query set size (right panel), we used a factor of $n + 2m \log n$ for normalization, motivated by the first upper bound in Theorem 4.2. (The factor 2 corresponds to an implementation detail.) Also, since k is comparable to n for the regime considered, the second upper bound in the theorem is much larger than the first. We did not use the query ratio since we are comparing networks of different sizes; larger the n, the lower is the query ratio. We observe that for some networks such as `astroph` and `facebook`, the number of queries is much better than the bound compared to other networks. Even among the three networks with comparable average degrees, we see that `lastfm` has a smaller normalized query set size compared to the other two networks. If two nodes share the same anchor node as well as a neighbor, then in a single query, this neighbor can be discovered for both nodes. This is partially captured by the clustering coefficient (CC) of the graph; larger the CC, higher is the number of such instances as we note in the result.

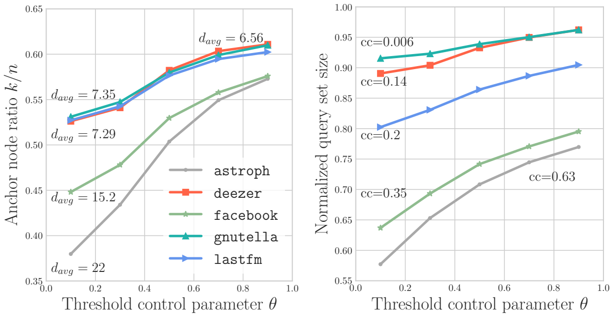

Figure 5. Experiments with real-world networks: The first two plots correspond to the anchor node ratio and the query ratio of ALG-DIR-SYM-BATCH on the considered real-world networks under random threshold assignment with varying θ values. In the second plot, the normalization factor is $n + 2m \log n$. In the plots, d_{avg} is average degree, while cc is the clustering coefficient.

Ordering of nodes and the number of anchor nodes. In all the experiments discussed above, we used one random node ordering per network. A natural question that arises is that how much is k dependent on this ordering. Since we do not assume any knowledge of the network other than the fact that it has n nodes, we studied how k varies across random permutations. In our experiments, we considered 100 random permutations for each network-θ instance. The results are in Figure 6. Our results show that the variance is generally small. In particular, it is much smaller in the case of the considered real-world networks than for the ER and BA networks. Our results indicate that simple extensions of our adaptive algorithms such as sampling ℓ node permutations randomly and using the permutation that leads to the smallest k might not be useful, as for each permutation, we will have to use n additional queries.

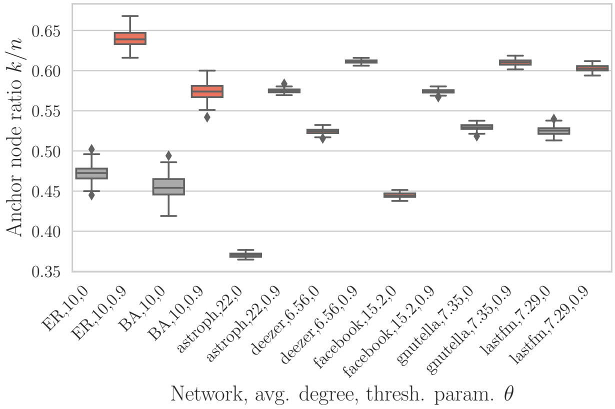

Figure 6. Random permutations of nodes and k: The variation in the number of anchor nodes k across 100 permutations. Two values of the threshold parameter are considered: $\theta = 0, 0.9$. All networks have comparable average degree.

8. Summary and Future Work

We considered the problem of inferring the topology and the local functions of a SyDS using active queries. We examined both batch and adaptive query modes. We showed that for certain classes of local functions, this inference can be carried out efficiently. Further, we established a lower bound on the size of any batch query set for inferring the topology and the local functions of threshold-SyDSs. We experimented with our algorithms using both synthetic and real-world social networks.

There are many directions for future work. One direction is to develop learning algorithms for SyDSs whose local functions are from other classes of Boolean functions. A second direction to devise learning algorithms that can reduce the number of queries when additional information about the underlying graph (e.g., degrees of all the nodes, properties such as acyclicity) is available. Finally, it will also be of interest to improve the lower bounds on the number of queries for various restricted classes of SyDSs.

Acknowledgments

We sincerely thank the ICML 2022 reviewers for providing very helpful comments. This research is supported by University of Virginia Strategic Investment Fund award number SIF160, Virignia Department of Health grant VDH-21-501-0135-1, NSF Grants OAC-1916805 (CINES), CCF-1918656 (Expeditions), IIS-1931628, IIS-1955797, and NIH grant R01GM109718.

References

Abasi, H., Bshouty, N. H., and Mazzawi, H. On exact learning monotone DNF from membership queries. *CoRR*, abs/1405.0792:1–16, 2014.

Abouheaf, M. I., Lewis, F. L., Vamvoudakis, K. G., Haesaert, S., and Babuska, R. Multi-agent discrete-time graphical games and reinforcement learning solutions. *Automatica*, 50(12):3038–3053, 2014.

Abrahao, B., Chierichetti, F., Kleinberg, R., and Panconesi, A. Trace complexity of network inference. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 491–499. ACM, 2013.

Adiga, A., Kuhlman, C. J., Marathe, M. V., Ravi, S. S., Rosenkrantz, D. J., and Stearns, R. E. Inferring local transition functions of discrete dynamical systems from observations of system behavior. *Theor. CS.*, 679:126–144, 2017.

Adiga, A., Kuhlman, C. J., Marathe, M. V., Ravi, S. S., Rosenkrantz, D. J., and Stearns, R. E. Learning the behavior of a dynamical system via a "20 questions" approach. In *Thirty second AAAI Conference on Artificial Intelligence*, pp. 4630–4637, 2018.

Adiga, A., Kuhlman, C. J., Marathe, M., Ravi, S. S., and Vullikanti, A. PAC learnability of node functions in networked dynamical systems. In *Proc. ICML 2019*, pp. 82–91, 2019.

Adiga, A., Kuhlman, C. J., Marathe, M., Ravi, S. S., Rosenkrantz, D. J., Stearns, R. E., and Vullikanti, A. Bounds and complexity results for learning coalition-based interaction functions in networked social systems. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI*, pp. 3138–3145. AAAI Press, 2020.

Angluin, D. and Slonim, D. K. Randomly fallible teachers: Learning monotone DNF with an incomplete membership oracle. *Machine Learning*, 14(1):7–26, 1994.

Berestovsky, N. and Nakhleh, L. An evaluation of methods for inferring Boolean networks from time-series data. *PLoS One*, 8:e66031-1–e66031-9, 2013.

Bonabeau, E. Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the national academy of sciences*, 99(suppl 3):7280–7287, 2002.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms*. MIT Press and McGraw-Hill, Cambridge, MA, Second edition, 2009.

Crama, Y. and Hammer, P. L. *Boolean Functions: Theory, Algorithms, and Applications*. Cambridge University Press, New York, NY, 2011.

Dasgupta, S., Hsu, D., Poulis, S., and Zhu, X. Teaching a black-box learner. In *International Conference on Machine Learning*, pp. 1547–1555. PMLR, 2019.

Eubank, S., Guclu, H., Kumar, V. A., Marathe, M. V., Srinivasan, A., Toroczkai, Z., and Wang, N. Modelling disease outbreaks in realistic urban social networks. *Nature*, 429 (6988):180–184, 2004.

Goldman, S. A. and Kearns, M. J. On the complexity of teaching. *Journal of Computer and System Sciences*, 50 (1):20–31, 1995.

Gomez-Rodriguez, M., Leskovec, J., and Krause, A. Inferring networks of diffusion and influence. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1019–1028. ACM, 2010.

González-Bailón, S., Borge-Holthoefer, J., Rivero, A., and Moreno, Y. The dynamics of protest recruitment through an online network. *Scientific Reports*, 1:7 pages, 2011.

Granovetter, M. Threshold models of collective behavior. *American Journal of Sociology*, pp. 1420–1443, 1978.

He, X., Xu, K., Kempe, D., and Liu, Y. Learning influence functions from incomplete observations. In *Advances in Neural Information Processing Systems*, pp. 2073–2081, 2016.

Hellerstein, L. and Servedio, R. A. On PAC learning algorithms for rich Boolean function classes. *Theoretical Computer Science*, 384(1):66–76, 2007.

Kauffman, S., Peterson, C., Samuelsson, B., and Troein, C. Random Boolean network models and the yeast transcriptional network. *Proc. National Academy of Sciences (PNAS)*, 100(25):14796–14799, Dec. 2003.

Kearns, M., Mansour, Y., Ron, D., Rubinfeld, R., Schapire, R., and Sellie, L. On the learnability of discrete distributions. In *Proc. ACM STOC*, pp. 273–282, 1994.

Kearns, M. J. and Vazirani, V. V. *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, MA, 1994.

Kempe, D., Kleinberg, J., and Tardos, E. Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 137–146, 2003.

Kleinberg, J., Mullainathan, S., and Ugander, J. Comparison-based choices. In *Proceedings of the 2017 ACM Conference on Economics and Computation*, pp. 127–144. ACM, 2017.

Laubenbacher, R. and Stigler, B. Design of experiments and biochemical network inference. In *Algebraic and Geometric Methods in Statistics*, pp. 1–13, 2009.

Liśkiewicz, M., Lutter, M., and Reischuk, R. Proper learning of k-term DNF formulas from satisfying assignments. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:114, 2017.

Lum, K., Swarup, S., Eubank, S., and Hawdon, J. The contagious nature of imprisonment: an agent-based model to explain racial disparities in incarceration rates. *Journal of The Royal Society Interface*, 11(98):2014.0409, 2014.

Macy, M. W. and Willer, R. From factors to actors: Computational sociology and agent-based modeling. *Annual Review of Sociology*, 28:143–166, 2002.

Mortveit, H. and Reidys, C. *An Introduction to Sequential Dynamical Systems*. Springer Science & Business Media, New York, NY, 2007.

Murphy, K. P. Passively learning finite automata. Technical Report 96-04-017, Santa Fe Institute, Santa Fe, NM, 1996.

Namatame, A. and Chen, S.-H. *Agent-based modeling and network dynamics*. Oxford University Press, 2016.

Narasimhan, H., Parkes, D. C., and Singer, Y. Learnability of influence in networks. In *Advances in Neural Information Processing Systems*, pp. 3186–3194, 2015.

Ogihara, M. and Uchizawa, K. Computational complexity studies of synchronous Boolean finite dynamical systems on directed graphs. *Inf. Comput.*, 256:226–236, 2017.

Ogihara, M. and Uchizawa, K. Synchronous Boolean finite dynamical systems on directed graphs over XOR functions. In *Proc. 45th MFCS*, pp. 76:1–76:13, Online publisher, 2020. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Romero, D., Meeder, B., and Kleinberg, J. Differences in the mechanics of information diffusion across topics: Idioms, political hashtags, and complex contagion on twitter. In *Proceedings of the 20th international conference on World wide web*, pp. 695–704. ACM, 2011.

Soundarajan, S. and Hopcroft, J. E. Recovering social networks from contagion information. In *Proceedings of the 7th Annual Conference on Theory and Models of Computation*, pp. 419–430. Springer, 2010.

Technical Supplement

Paper title: Efficiently Learning the Topology and Behavior of Networked Dynamical Systems via Active Queries

A. Additional Material for Section 4

A.1. Statement and Proof of Theorem 4.1:

Statement of Theorem 4.1: Let $\mathcal{S}$ be a symmetric-SyDS on a *directed* graph with node set $V = \{v_1 \ldots, v_n\}$. Then in polynomial time, a batch query set Q with $O(n^2)$ queries can be constructed, such that Q can be used to identify the local functions of $\mathcal{S}$ and for each node v whose local function is not a constant function, the set of edges entering v.

Proof: We recall from Figure 2 that the query set $\mathbf{Q} = \mathbf{Q}^1 \cup \mathbf{Q}^2 \cup \mathbf{Q}^3$ is constructed as follows.

1. $\mathbf{Q}^1$ has $n+1$ queries denoted by Q_i^1, $0 \leq i \leq n$. In query Q_0^1, all the n bits are 0. For $1 \leq i \leq n$, in query Q_i^1, the first i bits are 1 and the rest (if any) are 0.
2. $\mathbf{Q}^2$ has $n(n-1)/2$ queries denoted by $Q_{i,j}^2$, where $2 \leq i \leq n$ and $1 \leq j \leq i-1$. In query $Q_{i,j}^2$, the first i bits, *except* for bit j, are 1; bit j and all the remaining bits are 0.
3. $\mathbf{Q}^3$ has $n(n-1)/2$ queries denoted by $Q_{i,j}^3$, where $1 \leq i \leq n-1$ and $i+1 \leq j \leq n$. In query $Q_{i,j}^3$, the first $i-1$ bits and the bit j are 1; all the remaining bits are 0.

Let $\mathbf{R} = \mathbf{R}^1 \cup \mathbf{R}^2 \cup \mathbf{R}^3$ denote the responses obtained from the system.

First note that for any node v_p, since f_p is a symmetric function, f_p is a constant function iff the responses to all queries in $\mathbf{Q}^1$ have the same value of node v_p. So, if for all i, $0 < i \leq n$, $R_i^1[v_p]$ = $R_0^1[v_p]$, f_p is the constant function with value $R_0^1[v_p]$, and the incoming edges to v_p cannot be identified (as discussed in Section 2.3).

Otherwise, let r be the smallest integer such that $R_r^1[v_p] \neq R_0^1[v_p]$. As discussed in Section 3, node v_r is the **anchor node** for v_p. Since $R_r^1[v_p] \neq R_0^1[v_p]$ and configurations Q_{r-1}^1 and Q_r^1 differ only in the value of node v_r, v_r is one of the variables of the function f_p. Thus $v_r \in N[v_p]$, the closed neighborhood of v_p, and score$[Q_r^1, v_p]$ = score$[Q_{r-1}^1, v_p] + 1$. Further, if $r \neq p$, then there is an incoming edge from v_r to v_p. This establishes the correctness of Step 3(b) of the algorithm in Figure 2.

Those queries in $\mathbf{Q}^2$ that are of the form $Q_{r,j}^2$ are used to identify those nodes v_j with $j < r$ such that there is an incoming edge from v_j to v_p. Those queries in $\mathbf{Q}^3$ that are of the form $Q_{r,j}^3$ are used to identify those nodes v_j with $j > r$ such that there is an incoming edge from v_j to v_p. We now show that the algorithm correctly identifies these edges.

For each node v_j, where $j < r$ and $j \neq p$, consider the query $Q_{r,j}^2$. Configurations Q_r^1 and $Q_{r,j}^2$ differ only in the value of node v_j. If there is an edge from v_j to v_p, then score$[Q_{r,j}^2, v_p]$ = score$[Q_{r-1}^1, v_p]$, so $R_{r,j}^2[v_p] = R_{r-1}^1[v_p] = R_0^1[v_p]$. If there is no edge from v_j to v_p, then score$[Q_{r,j}^2, v_p]$ = score$[Q_r^1, v_p]$, so $R_{r,j}^2[v_p] = R_r^1[v_p] \neq R_0^1[v_p]$. Thus, the value of $R_{r,j}^2[v_p]$ indicates whether of not there is an incoming edge from v_j to v_p. This establishes the correctness of Step 3(c) of the algorithm in Figure 2.

For each node v_j, where $j > r$ and $j \neq p$, consider the query $Q_{r,j}^3$. Configurations Q_{r-1}^1 and $Q_{r,j}^3$ differ only in the value of node v_j. If there is no edge from v_j to v_p, then score$[Q_{r,j}^3, v_p]$ = score$[Q_{r-1}^1, v_p]$, so $R_{r,j}^3[v_p] = R_{r-1}^1[v_p] = R_0^1[v_p]$. If there is an edge from v_j to v_p, then score$[Q_{r,j}^3, v_p]$ = score$[Q_r^1, v_p]$, so $R_{r,j}^3[v_p] = R_r^1[v_p] \neq R_0^1[v_p]$. Thus, the value of $R_{r,j}^3[v_p]$ indicates whether of not there is an incoming edge from v_j to v_p. This establishes the correctness of Step 3(d) of the algorithm in Figure 2.

We now argue that the responses to the queries in $\mathbf{Q}^1$ can be used to identify the local (symmetric) functions of nodes whose local function is not a constant function. Consider a given node v_p whose local function f_p is not a constant function. We now describe how to construct the table T_p that specifies symmetric function f_p. Let $d(p)$ denote the indegree of node v_p. Table T_p has an entry, $T_p[j]$, for each j such that $0 \leq j \leq d(p)+1$. For each j, $0 \leq j \leq d(p)+1$, let k be the minimum value such that Q_k^1 has exactly j nodes from $N[v_p]$. Thus, the value $R_k^1[v_p]$ gives the value of the function f_p when the input has exactly j ones. In other words, $T_p[j] = R_k^1[v_p]$. This establishes the correctness of Step 5 of the algorithm in Figure 2. ■

A.2. An Adaptive Inference Algorithm for Symmetric-SyDSs

Here, we present our algorithm for inferring symmetric-SyDSs under the adaptive query mode. We present the discussion for directed graphs. The modifications needed for undirected graphs are similar to those discussed at the end of Section 4.1.

Theorem A.1. *There is an algorithm under the adaptive query mode that can learn the local functions of any symmetric-SyDS whose underlying graphs are directed. For those nodes whose local functions are not constant func-*

tions, the algorithm identifies the incoming edges. If the SyDS has n nodes and m edges, the number of queries used by the algorithm is $O(n + m\log(n))$.

Proof: Let $V = \{v_1, \ldots, v_n\}$ be the given node set of $\mathcal{S}$.

The adaptive scheme begins with the set of $n+1$ queries $\mathbf{Q}^1$ from Algorithm ALG-DIR-SYM-BATCH of Figure 2:

$\mathbf{Q}^1 = \{Q_i^1, 0 \leq i \leq n$, where for each k, $1 \leq k \leq n$, $Q_i^1[v_k] = 1$ iff $k \leq i\}$.

For each node $v_p \in V$, the algorithm identifies node function f_p, and, when f_p is not a constant function, the incoming edges of v_p, as follows.

Suppose that the value of v_p in the responses to the all the queries in $\mathbf{Q}^1$ is the same, i.e., for all i, $0 < i \leq n$, $R_i^1[v_p] = R_0^1[v_p]$. Then f_p is the constant function with value $R_0^1[v_p]$, and the incoming edges to v_p cannot be identified.

Otherwise, we generalize the concept of an anchor node to a set of anchor nodes, as follows. We call each node v_r such that $R_r^1[v_p] \neq R_{r-1}^1[v_p]$ a **anchor node** for v_p. For each **anchor node** v_r for v_p, configurations Q_{r-1}^1 and Q_r^1 differ only in the value of node v_r; thus, v_r is one of the variables of function f_p. Hence, $v_r \in N[v_p]$, the closed neighborhood of v_p, and $\texttt{score}[Q_r^1, v_p] = \texttt{score}[Q_{r-1}^1, v_p] + 1$. If $r \neq p$, then there is an incoming edge from v_r to v_p.

We define a **run** for v_p to be the sequence of nodes before the the first anchor node for v_p, between two consecutive anchor nodes, or after the last anchor node. Thus, if there are k anchor nodes for v_p, then there are $k+1$ runs. Each node in a run is potentially a member of $N[v_p]$.

Let W denote a nonempty sequence of consecutive nodes within some run. We will define a query Q_W^p that we will show can be used to determine whether W contains a member of $N[v_p]$.

Suppose W occurs in the first run, i.e., before the first anchor node for v_p. Let v_r denote this first anchor node. Then Q_W^p is constructed as follows.

For each k, $1 \leq k \leq n$, $Q_W^p[v_k] = 1$ iff $k \leq r$ and $k \notin W$.

Note that configurations Q_r^1 and Q_W^p differ only in the value of the nodes in W. If W contains at least one member of $N[v_p]$, then $\texttt{score}[Q_W^p, v_p] < \texttt{score}[Q_r^1, v_p]$, so $Q_W^p[v_p] = R_0^1[v_p]$. If W contains no member of $N[v_p]$, then $\texttt{score}[Q_W^p, v_p] = \texttt{score}[Q_r^1, v_p]$, so $Q_W^p[v_p] = R_r^1[v_p] \neq R_0^1[v_p]$. Thus, $Q_W^p[v_p] = R_0^1[v_p]$ iff W contains at least one member of $N[v_p]$.

Suppose W occurs in a later run, i.e., after at least one anchor node for v_p. Let v_r denote the anchor node immediately preceding the run containing W. Then Q_W^p is constructed as follows.

For each k, $1 \leq k \leq n$, $Q_W^p[v_k] = 1$ iff $k < r$ or $k \in W$.

Note that configurations Q_r^1 and Q_W^p differ only in the value of v_r and the nodes in W. Suppose that W contains no member of $N[v_p]$, then $\texttt{score}[Q_W^p, v_p] = \texttt{score}[Q_{r-1}^1, v_p]$, so $Q_W^p[v_p] = R_{r-1}^1[v_p] \neq R_r^1[v_p]$. However, suppose that W contains at least one member of $N[v_p]$. Let $v_{r'}$ be the last node in the run containing W. Then $\texttt{score}[Q_r^1, v_p] \leq \texttt{score}[Q_W^p, v_p] < \texttt{score}[Q_{r'}^1, v_p]$, so $Q_W^p[v_p] = R_r^1[v_p]$. Thus, $Q_W^p[v_p] = R_r^1[v_p]$ iff W contains at least one member of $N[v_p]$.

In summary, query Q_W^p can be used to determine whether W contains a member of $N[v_p]$.

The adaptive query algorithm searches each run, finding all the members of $N[v_p]$ in the run, as follows. Consider a given run. The algorithm executes a loop, each iteration of which searches for the next member of $N[v_p]$ in the run. In each iteration of this loop, we let W denote a set of **candidate nodes** in the run, which are potentially members of $N[v_p]$. Initially, W is the set of all nodes in the run.

Now consider a given iteration of the loop. If W is empty, it contains no members of $N[v_p]$, so the loop terminates. So suppose that W is nonempty. The algorithm constructs the query Q_W^p. If the response to this query indicates that W contains no members of $N[v_p]$, then the loop terminates. Otherwise, the algorithm performs a binary search on W to identify the leftmost member of W that is in $N[v_p]$. Once this node has been found, this node and all the candidate nodes to its left are deleted from W, and the remaining members of W constitute a modified set of candidate nodes. The next iteration proceeds with this new value of W. ■