# PAC Learnability of Node Functions in Networked Dynamical Systems

Abhijin Adiga [1] Chris J. Kuhlman [1] Madhav V. Marathe [1 2] S. S. Ravi [1 3] Anil K. Vullikanti [1 2]

## Abstract

We consider the PAC learnability of the functions at the nodes of a discrete networked dynamical system, assuming that the underlying network is known. We provide tight bounds on the sample complexity of learning threshold functions. We establish a computational intractability result for efficient PAC learning of such functions. We develop efficient consistent learners when the number of negative examples is small. Using synthetic and real-world networks, we experimentally study how the network structure and sample complexity influence the quality of inference.

## 1. Introduction

Many real world phenomena (disease, influence and social behavior, neuronal activity, magnetic systems) can be formally represented as networked discrete dynamical systems (Valente, 1996; Schelling, 1978; Amini & Fountoulakis, 2014). Many such systems that include popular models such as SEIR (Newman, 2002) and Linear Threshold (Kempe et al., 2003) fall into the generic class of **Synchronous dynamical systems** (SyDSs). A SyDS consists of a graph whose nodes represent entities and whose edges represent interactions among the entities. Nodes have states and a local function at each node determines the next state of the node using the current states of the node and its neighbors. In a SyDS, the propagation of a contagion evolves in discrete time steps. In practice, some components of a SyDS are unknown; learning them is an active area of research. Some of this research is based on observing the system (Adiga et al., 2017; González-Bailón et al., 2011; Narasimhan et al., 2015; Kempe et al., 2003; Lokhov, 2016) while others rely on active interactions with the system in the form of queries (Bei et al., 2016; Kleinberg et al., 2017; Adiga et al., 2018).

[1]Biocomplexity Institute and Initiative, University of Virginia, Charlottesville, VA, USA. [2]Also with the Department of Computer Science, University of Virginia, Charlottesville, VA, USA. [3]Also with the Department of Computer Science, University at Albany – SUNY, Albany, NY, USA. Correspondence to: Abhijin Adiga <abhijin@virginia.edu>.

*Proceedings of the 36th International Conference on Machine Learning*, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

We consider the problem of learning the node functions of a SyDS with a known graph and binary node states under the probably approximately correct (PAC) learning model pioneered by Valiant (1984). The PAC model has been applied to many different learning contexts such as learning classes of Boolean functions, half spaces, automata, etc. (Kearns & Vazirani, 1994). While PAC learnability has been studied for classes of individual Boolean functions (e.g., (Hellerstein & Servedio, 2007)), there has been limited work on dynamical systems, which can be viewed as Boolean functions that are connected through a network. Only recently, it has been applied in the context of learning influence functions of nodes in stochastic networked dynamical systems (Narasimhan et al., 2015; He et al., 2016).

**Threshold functions** are a commonly used class of local functions in dynamical system models (Granovetter, 1978; Watts, 2002). A $t$-threshold function is a Boolean function whose value is 1 iff at least $t$ of its inputs have value 1. Variants of this model include bithreshold (Kuhlman et al., 2011) and progressive threshold (Amini & Fountoulakis, 2014). Our focus is on learning this class of functions (i.e., the threshold value of each node in the network) under the PAC model. Let $n$ be the number of nodes in the graph. A **configuration** $(s_1, s_2, \ldots, s_n)$ specifies the state of each node at a certain time. Given a configuration $C_1$ at time $\tau$, the configuration $C_2$ at time $\tau+1$ is called the **successor** of $C_1$. Each example given to the learner is a pair of configurations $(C_1,\ C_2)$. An example is labeled 'positive' if $C_2$ is the successor of $C_1$; otherwise, it is labeled 'negative'. Under the PAC model, these examples are drawn from an unknown distribution.

**Summary of results.** Our focus is on SyDSs, where the local function at each node is a threshold function; that is, the hypothesis space consists of $n$-tuples of the form $(t_1, t_2, \ldots, t_n)$, where each $t_i$, a non-negative integer, is the threshold value assigned to node $v_i$, $1 \le i \le n$. Following Mitchell (1997), we use the phrase "sample complexity", represented by $\mathcal{M}(\epsilon, \delta)$, to denote the number of examples needed under the PAC model to learn a hypothesis space for given pair of error and probability values $(\epsilon, \delta)$. We assume that $\epsilon, \delta \in (0, \frac{1}{2})$. Our results are summarized below.

1. **Upper bounds**: For learning the hypothesis space of

threshold functions for SyDSs, we show that $\mathcal{M}(\epsilon, \delta)$ is at most $\frac{1}{\epsilon}\big(n\log(d_{\text{avg}}+3)+\log(1/\delta)\big)$ where $d_{\text{avg}}$ is the average node degree of the underlying graph. We also extend this to other classes of threshold functions.

2. **Lower bounds**: We prove that the Vapnis-Chervonenkis (VC) dimension of the hypothesis space of threshold functions is at least $\lfloor n/4 \rfloor$ for *any* graph. From well known results in computational learning theory (Kearns & Vazirani, 1994), these lower bounds on the VC dimension imply that the sample complexity for learning threshold functions is $\Omega(n/\epsilon)$. Since $d_{\text{avg}} \leq n-1$, for any fixed $\delta$, the upper bound on the sample complexity mentioned in Item 1 above is within a factor $O(\log n)$ of the lower bound. Further, for graphs with constant average degree, this factor is $\Theta(1)$. As a special case, when the underlying graph is a clique on $n$ nodes, we show that the VC dimension of the hypothesis space is at most $n+1$. In contrast, most prior work on PAC learnability of graph dynamical system properties has been restricted to showing polynomial upper bounds on the sample complexity, e.g., (Narasimhan et al., 2015).

3. **Hardness of learning**: Even though there is a polynomial upper bound on the sample complexity, we show that when there are positive and negative examples, the hypothesis class of threshold functions is not *efficiently* PAC learnable, unless the complexity classes **NP** and **RP** (Randomized Polynomial time) coincide. Such complexity results are known in the learning theory literature for several other problems such as learning $k$-term DNF, neural networks, etc. (Kearns & Vazirani, 1994).

4. **Efficient learning algorithms**: For the case when there are only positive examples, we present an algorithm which learns the thresholds in time $O(|\mathcal{E}|n)$, where $\mathcal{E}$ is the set of examples and $n$ is the number of nodes. Further, when a set $\mathcal{E}_N$ of negative examples is also given, we present a dynamic programming algorithm that learns in time $O(2^{|\mathcal{E}_N|}\text{poly}(n))$, which is polynomial when $|\mathcal{E}_N| = O(\log n)$. Also, using submodular function maximization under matroid constraints, we present an efficient learner which is consistent with all the positive examples and at least $(1-1/e)$ fraction of the negative examples. These results show that computational intractability arises in this case when the number of negative examples is large.

5. **Experiments**: We present experimental results using both synthetic and real-world networks to demonstrate how network structure and sample complexity influence the quality of the inferred system. We also provide experimental results that interpolate between theoretical results of limiting cases: as graph density decreases from a fully connected graph to a sparse graph, differences between the true and inferred systems first increase, and then decrease, thus exhibiting non-linear and non-monotonic behavior. Another interesting finding is how, in learning the same dynamical system, differences in the distributions of configurations can lead to widely different qualities of inferred systems.

Due to limited space, many proofs and additional experimental results appear in (Adiga et al., 2019).

**Related work.** Inferring properties of networked dynamical systems from time-series data of node activations is a popular topic. Brugere et al. (2018) and Guille et al. (2013) provide comprehensive surveys of the literature on inferring networks and propagation model parameters from information or infection cascades. González-Bailón et al. (2011) present techniques for learning thresholds of nodes in a Twitter network using data from retweets. In contrast, PAC learnability of networked dynamical system is an emerging area of research. Recently, Narasimhan et al. (2015) and He et al. (2016) studied the PAC learnability of the influence function of popular stochastic propagation models – independent cascade and linear threshold from complete and partial observations. Lokhov (2016) uses a dynamic message-passing algorithm to reconstruct parameters of a spreading model given infection cascades.

There has been extensive research on PAC learning threshold functions and in general, Boolean functions. Hellerstein & Servedio (2007) provide a survey covering learnability of halfspaces, polynomial threshold functions, decision trees and disjunctive normal form (DNF) formulas. Learning the local function of a single vertex of a threshold SyDS is a special case of learning halfspaces (Blumer et al., 1989) with all weights equal to 1. Our results on learning threshold functions of SyDSs under the PAC model show that the network structure plays an important role in determining the sample complexity.

Recently, Adiga et al. (2017; 2018) considered the problem of inferring threshold SyDS, but for different models of observation. While Adiga et al. (2017) devise algorithms for learning thresholds of a dynamical system using information about the system's trajectories, Adiga et al. (2018) study inference under active querying, where the user may ask for the successor of an arbitrary configuration.

## 2. Preliminaries

**The dynamical system model.** 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 graph with $|V| = n$, 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 $v_i$, $1 \leq i \leq n$. At any time, each node of $G$ has a state value from $\mathbb{B}$. 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 $\tau$, if $s_i^\tau \in \mathbb{B}$ is the state of node $v_i$ ($1 \leq i \leq n$), the **configuration** $C$ of the SyDS is the $n$-vector $(s_1^\tau, s_2^\tau, \ldots, s_n^\tau)$. The system evolves in discrete time steps by the repeated application of $\mathcal{F}$. If a SyDS has a one step transition from configuration $C_1$ to $C_2$, then $C_2$ is the **successor** of $C_1$. In this paper, the local functions considered are deterministic, and therefore, the successor of each configuration is *unique*.

**Threshold functions.** The local function $f_v$ associated with node $v$ of a SyDS $\mathcal{S}$ is a $t_v$-**threshold** function for some integer $t_v \geq 0$ if the following condition holds: the value of $f_v$ is 1 if the number of 1's in the input to $f_v$ is *at least* $t_v$; otherwise, the value is 0. Using $d(v)$ to denote the degree of node $v$, the number of inputs to the function $f_v$ is $d(v)+1$. We assume that $0 \leq t_v \leq d(v)+2$. The threshold values 0 and $d(v)+2$ give rise to local functions that always output 1 and 0 respectively. Given a configuration $C$ and a node $v$, the **score** of configuration $C$ with respect to $v$, score$(C, v)$, is the number of 1's in the input provided by $C$ to the local function at $v$.

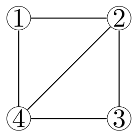


*Figure 1.* An example of a SyDS. The thresholds are $t_1 = 1$, $t_2 = 1$, $t_3 = 2$ and $t_4 = 3$.

**Example:** We present the graph of a threshold SyDS in Figure 1. The initial configuration is $(0,0,0,1)$. It can be verified that the system goes through the following sequence of configurations during the next three time steps: $(0,0,0,1) \rightarrow (1,1,0,0) \rightarrow (1,1,1,0) \rightarrow (1,1,1,1)$.

**Positive and negative examples.** The goal is to learn the concept class of threshold functions for SyDSs whose underlying graphs are known. Thus, given an $n$-node SyDS, the hypothesis space $H$ for the concept class contains $n$-tuples of the form $(t_1, t_2, \ldots, t_n)$, where each $t_i$ is the threshold value assigned to node $v_i$. Each example $\eta$ given to learner is a pair of configurations $(C_1,\ C_2)$ of the SyDS. We use $\mathcal{E}$ to denote the set of examples given to a learner. An example is labeled 'positive' if $C_2$ is the successor of $C_1$ and 'negative' otherwise. Thus, the instance space from which examples are drawn consists of pairs of configurations, with an appropriate label for each example. Positive examples can be generated by observing or querying a system (Adiga et al., 2017; 2018). A negative example can be constructed from a positive example $(C_1, C_2)$ by randomly complementing one or more bits of $C_2$. The positive and negative examples are similar to membership queries used to learn Boolean functions (e.g., (Angluin & Slonim, 1994)).

**Some PAC model definitions.** We will assume that the reader is familiar with the basic concepts of the PAC model such as efficient PAC learnability covered in many texts (e.g., (Kearns & Vazirani, 1994; Mitchell, 1997)). The **true error** (denoted commonly by $\text{error}_{\mathcal{D}}(h)$) of a hypothesis $h$ is the probability that $h$ will misclassify an example chosen at random using a distribution $\mathcal{D}$; that is, $\text{error}_{\mathcal{D}}(h) = \Pr_{x \in \mathcal{D}}[c(x) \neq h(x)]$, where $c$ is the target concept. For given values of $\epsilon$ and $\delta$, the **sample complexity** of a learner is the number of examples needed by the learner to output an appropriate hypothesis $h$. We denote this quantity by $\mathcal{M}(\epsilon, \delta)$. Below is a well-known upper bound on $\mathcal{M}(\epsilon, \delta)$ established in (Haussler, 1988) based on the size of the hypothesis space $H$.

$$\mathcal{M}(\epsilon, \delta) \leq \frac{1}{\epsilon}\big(\log|H| + \log(1/\delta)\big)\ . \qquad (1)$$

We also use the concept of Vapnik-Chervonenkis (VC) dimension (Kearns & Vazirani, 1994). Given a set $\mathcal{E}$ of labeled examples, we say that a hypothesis $h$ (i.e., a threshold assignment to the the nodes of the SyDS) from the hypothesis space $H$ is **consistent** with $\mathcal{E}$, if $h$ correctly classifies each example in $\mathcal{E}$. Given a set $\mathcal{E}$ of unlabeled examples from the instance space, a **dichotomy** of $\mathcal{E}$ partitions $\mathcal{E}$ into two subsets $\mathcal{E}_P$ and $\mathcal{E}_N$ of positive and negative examples respectively. A set of examples $\mathcal{E}$ is **shattered** by the hypothesis space $H$ if for every dichotomy of $\mathcal{E}$, there is a hypothesis $h \in H$ that is consistent with the labeled examples generated by the dichotomy. The **VC dimension** $\Delta$ of $H$ defined over the instance space is the largest finite subset of the instance space that can be shattered by $H$. Since our hypothesis space $H$ is finite, the VC dimension of $H$ is also finite. Below is a result from (Hanneke, 2016) on the bounds for sample complexity in terms of the VC dimension $\Delta$.

**Lemma 1.** *For any* $\epsilon \in (1, 1/8]$ *and* $\delta \in (0, 1/100]$, *sample complexity* $\mathcal{M}(\epsilon, \delta) = O\Big(\frac{1}{\epsilon}\big(\Delta\ \text{Log}(1/\epsilon) + \text{Log}(1/\delta)\big)\Big)$ *and* $\mathcal{M}(\epsilon, \delta) = \Omega\Big(\frac{1}{\epsilon}\big(\Delta + \text{Log}(1/\delta)\big)\Big)$, *where* $\text{Log}(z) = \log\big(\max(z, e)\big)$ *and* $e$ *is the base of common logarithm.* □

When $\epsilon$ and $\delta$ are fixed, the above lemma points out that for a hypothesis space with VC dimension $d$, the sample complexity is $\Theta(\Delta)$.

## 3. Bounds on the Sample Complexity

**Upper bound.** Here, we bound the size of the hypothesis space; the bound on sample complexity follows by a direct application of Inequality (1).

**Theorem 1.** *Let* $G(V, E)$ *be a graph with* $n$ *nodes and average degree* $d_{avg}$. *Let* $\epsilon, \delta > 0$ *be given. The sample complexity* $\mathcal{M}(\epsilon, \delta)$ *of threshold SyDS satisfies* $\mathcal{M}(\epsilon, \delta) \leq \frac{1}{\epsilon}\big(n \log(d_{avg} + 3) + \log(1/\delta)\big)$.

**Proof:** Each node $v$, with degree $d(v)$, can be assigned a threshold in the range 0 to $d(v)+2$. So, for each node, there are $d(v) + 3$ possible thresholds. Therefore, $|H| = \prod_{v\in V}(d(v) + 3) \leq \left(\frac{1}{n}\sum_{v\in V}\left(d(v) + 3\right)\right)^n = \left(d_{\text{avg}} + 3\right)^n$. The inequality follows from arithmetic mean–geometric mean inequality. The upper bound on $\mathcal{M}(\epsilon, \delta)$ follows by applying Equation (1). □

In (Adiga et al., 2019), we show similar results for SyDSs with bithreshold and progressive threshold functions.

Theorem 1 corresponds to an extreme case where no additional information on the threshold functions is known. There are at least two factors that influence the size of the hypothesis space. Firstly, if local functions of different vertices are correlated and the relationship is known, then determining one threshold value would lead to estimates of others. Such models have been considered in disease dynamics (Miller, 2009). Secondly, any additional information about the local function restricts the number of hypotheses possible (e.g. (Romero et al., 2011)). To elaborate on this point, we consider the following setting. Let $\{V_1, V_2, \ldots, V_k\}$ be a partition of the node set $V$. Every node in $V_i$ has the same threshold $t_i \in T_i$, where $T_i$ is a subset of threshold values. Then, the number of possible threshold assignments is at most $\prod_{i=1}^{k}|T_i|$ and therefore, from Equation (1), $\mathcal{M}(\epsilon, \delta) \leq \frac{1}{\epsilon}\left(\sum_{i=1}^{k}\log|T_i| + \log(1/\delta)\right)$. Theorem 1 is the limiting case with $k = n$ and $T_i = \{0, \ldots, d(v_i) + 2\}$.

**Lower bound.** We will establish a lower bound on the VC dimension of the hypothesis class and then apply Lemma 1 to obtain a lower bound on the sample complexity. We note that these results hold for the class of bithreshold SyDSs since threshold SyDSs are contained in this class.

**Theorem 2.** *For a threshold SyDS defined on a graph $G(V, E)$ with $n$ nodes, the VC dimension of the hypothesis space is $\geq \lfloor n/4 \rfloor$. Hence, $\mathcal{M}(\epsilon, \delta) = \Omega(n)$ for constant $\epsilon$ and $\delta$.*

We need two lemmas to prove the above theorem.

**Lemma 2.** *Let $I$ be an independent set in graph $G(V, E)$. There exists a set of configuration–successor pairs of size $|I|$ that can be shattered.*

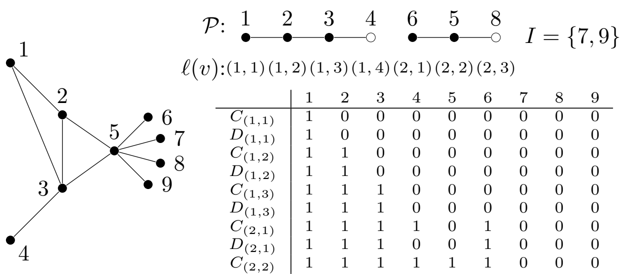
**Proof:** Let $v_1, v_2, \ldots, v_{|I|}$ denote an arbitrary ordering of the vertices of $I$, The remaining vertices are ordered arbitrarily following $v_{|I|}$. We construct a set $T$ with $|I|$ configuration–successor pairs as follows. Let $C_i$ be the configuration in which the state of $v_i$ is 1 and the states of all other nodes is 0, $1 \leq i \leq |I|$. Let $T = \{(C_i, C_i) : 1 \leq i \leq |I|\}$. We now show that $T$ can be shattered. Let $A \subseteq T$. Vertex thresholds are assigned as follows: If $(C_i, C_i) \in A$, then set $t(v_i) = 1$; else set $t(v_i) = 2$. If $(C_i, C_i) \in A$, then the successor of $C_i$ for the assigned thresholds is $C_i$ since $\text{score}(v_i, C_i) = t(v_i) = 1$ while for every other vertex $v_j$, $\text{score}(v_j, C_i) \leq 1 < t(v_j) = 2$. Suppose $(C_i, C_i) \notin A$; then, $\text{score}(v_i, C_i) = 1 < t(v_i) = 2$. Thus, the state of $v_i$ is 0 and therefore, the successor is not $C_i$. □

**Definition 1.** An **ordered path** in $G(V, E)$ is an ordered subset of vertices $P = (v_1, v_2, \ldots, v_k)$, where $k \geq 2$, such that edge $\{v_i, v_{i+1}\} \in E$ for $1 \leq i \leq k - 1$.

**Lemma 3.** *Let $\mathcal{P}$ be a collection of node-disjoint ordered paths in $G(V, E)$. Let $n'$ be the number of participating nodes in $\mathcal{P}$. Then, there exists a set of configuration–successor pairs of size at least $\lceil n'/2 \rceil$ that can be shattered.*

**Proof sketch:** We will describe the construction of the configuration–successor pairs. The proof that the set can be shattered is in (Adiga et al., 2019). We will arbitrarily order the paths in $\mathcal{P}$. Each node $v$ in $\mathcal{P}$ is associated with a tuple $\ell(v) = (o_v, i_v)$ referred to as its *label*, where $o_v$ (outer position) is the position of its path in the ordering of the paths and $i_v$ (inner position) is the node's position in the path. For any two distinct vertices $v$ and $v'$, $\ell(v') < \ell(v)$ if and only if (i) $o_{v'} < o_v$ or (ii) $o_{v'} = o_v$ and $i_{v'} < i_v$. Let $N(v)$ be the set of neighbors of $v$ (open neighborhood). For any $i$, define $d_{(o,i)}(v) = |N(v) \cap \{v' \mid \ell(v') \leq (o, i)\}|$.

Let $L$ be the set of labels $\ell = (o, i)$ such that the $i$th vertex is not the last node in the corresponding ordered path. For each $\ell \in L$, the configuration–successor pair $(C_\ell, D_\ell)$ is as follows: In $C_\ell$, all labeled nodes $v$ with $\ell(v) \leq \ell$ are set to state 1 and the rest are set to 0. In $D_\ell$, all $v$ such that $\ell(v) \in L$ and $\ell(v) \leq \ell$ are set to state 1 and the rest to state 0. All nodes not in $\mathcal{P}$ are also set to 0. The proof that the set $T = \{(C_\ell, D_\ell) \mid \ell \in L\}$ can be shattered is in (Adiga et al., 2019). See Figure 2 for an example. □



$\mathcal{P}$: 1 2 3 4 6 5 8
$I = \{7, 9\}$
$\ell(v)$: (1, 1) (1, 2) (1, 3) (1, 4) (2, 1) (2, 2) (2, 3)
1
2
3
4
5
6
7
8
9

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| $C_{(1,1)}$ | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $D_{(1,1)}$ | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $C_{(1,2)}$ | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $D_{(1,2)}$ | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $C_{(1,3)}$ | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| $D_{(1,3)}$ | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| $C_{(2,1)}$ | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| $D_{(2,1)}$ | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| $C_{(2,2)}$ | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| $D_{(2,2)}$ | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |

For $A = \{(C_{(1,1)}, D_{(1,1)}, (C_{(2,1)}, D_{(2,1)})\}$, threshold assignments are $t(1) = d_{(1,1)} + 1 = 1$; $t(2) = d_{(1,2)} + 2 = 3$; $t(3) = d_{(1,3)} + 2 = 4$; $t(6) = d_{(2,1)} + 1 = 1$; $t(5) = d_{(2,2)} + 2 = 4$; $t(4) = t(8) = t(7) = t(9) = n + 1 = 10$

*Figure 2.* Example of configuration–successor pair construction using ordered paths. Here, $T$ is the set of $(C_{(i,j)}, D_{(i,j)})$ pairs defined in the table and $A$ is a subset of $T$.

**Proof of Theorem 2:** We use a greedy strategy to obtain $\mathcal{P}$. Let $V' = V$. At the start of each iteration, remove a vertex from $v \in V'$ and set $P = \{v\}$. The current path has only $v$, which is an end point. The path is "grown" as follows. For an end point of the current path in $P$, pick a neighbor $v'$ from $V' \setminus P$ if one exists. Move $v'$ from $V'$ to $P$ and update the path with $v'$ as an end point and $v$ as an internal point if $|P| > 2$. The process terminates when no new neighbor can be added from $V'$ for the current path's end points. The path is ordered starting and ending with the current end points. $P$ is added to $\mathcal{P}_{\geq 3}$ (i.e., the subset of paths with $\geq 3$ nodes) and the process continues until $V'$ has no paths of length $\geq 2$. In that case, we set $V_{<3} = V'$. What remains of $V'$ induces an independent set of size at least $|V'|/2$. This is set to $I$. If $|I| \geq n/4$, then, the proof follows from Lemma 2. Otherwise, the number of vertices in $\mathcal{P}$ is at least $n/2$. In that case, the proof follows from Lemma 3. □

**A tighter bound for complete graphs.** Here, we observe that the relationship between sample complexity and edge density is not straightforward. When the underlying network $G$ is a complete graph, from Theorem 1, the sample complexity for the threshold SyDSs is $O(n \log n)$ for constant $\epsilon$ and $\delta$. Our next result shows that the sample complexity in this case is actually $O(n)$.

**Theorem 3.** *For a threshold SyDS defined on a complete graph $G$ of size $n$, the VC-dimension of the hypothesis space is at most $n + 1$.*

**Proof (idea):** We establish an upper bound on the VC dimension. The proof is based on pigeon-hole principle and appears in (Adiga et al., 2019). □

## 4. Efficiency of Learning Algorithms

We will show that the complexity of learning the thresholds hinges on the number of negative examples. We will also develop efficient exact and approximation algorithms. Given a collection $\mathcal{E}$ of labeled examples, a learner is **consistent** with respect to $\mathcal{E}$ if the hypothesis (a threshold assignment for all the nodes) produced by the learner satisfies *all* the examples in $\mathcal{E}$; that is, (a) for each positive example $(C_1,\ C_2)$, $C_2$ is the successor of $C_1$ and (b) for each negative example $(C_1,\ C_2)$, $C_2$ is *not* the successor of $C_1$.

**Complexity of learning from positive and negative examples.** Using a known approach from the literature (see e.g., (Kearns & Vazirani, 1994)), we show that if there is an efficient PAC learning algorithm with positive and negative examples, then there is a randomized polynomial time (**RP**-time) algorithm for every problem in **NP**, thus implying that **NP** = **RP**. To prove this result, we use an **NP**-complete problem defined in (Adiga et al., 2017). Recall that a configuration $C$ of a SyDS is a *fixed point* if the successor of $C$ is $C$ itself; otherwise, it is a non-fixed point.

**Problem 1. TA-FNF** (Adiga et al., 2017): Given the graph $G(V, E)$ of a SyDS $\mathcal{S}$ and two disjoint sets of configurations $F_1$ and $F_2$, is there a threshold assignment for the nodes of $\mathcal{S}$ such that each configuration in $F_1$ is a fixed point and each configuration in $F_2$ is *not* a fixed point?

Now, we state our complexity result.

**Theorem 4.** *Given a graph $G(V, E)$ and a set $\mathcal{E}$ of positive and negative examples, the concept class of threshold functions is not efficiently PAC learnable, unless **NP** = **RP**.*

**Proof sketch:** Suppose the concept class of threshold functions is efficiently PAC learnable from a set of positive and negative examples. Let $\mathcal{A}$ be an efficient learning algorithm whose running time is polynomial in the size of the problem instance and the values $1/\delta$ and $1/\epsilon$. We show that $\mathcal{A}$ can be used to devise an **RP** algorithm for TA-FNF. For brevity, we denote the learning problem by Pos-Neg-Learn. Given an instance $I$ of the TA-FNF problem, we construct an instance $I'$ of Pos-Neg-Learn as follows.

(a) The underlying graph $G(V, E)$ of the SyDS $\mathcal{S}$ for the Pos-Neg-Learn problem is the same as that of the TA-FNF problem.

(b) For each configuration $C \in F_1 \cup F_2$, we construct the example $(C, C)$. If $C \in F_1$, the example is labeled positive, else, it is labeled negative. This is the set of examples $\mathcal{E}$.

(c) Let $\epsilon = 1/(2|\mathcal{E}|)$ and $\delta = 0.1$.

Using the assumed efficient learning algorithm $\mathcal{A}$, we can now construct an algorithm $\mathcal{A}_1$ for TA-FNF as follows.

1. Run the learning algorithm $\mathcal{A}$ on instance $I'$. Whenever $\mathcal{A}$ requests an example, it is given one chosen uniformly randomly from $\mathcal{E}$. (Thus, the chosen distribution does not produce any examples that are not in $\mathcal{E}$.)

2. If $\mathcal{A}$ produces a threshold assignment that is consistent with all the examples in $\mathcal{E}$, then $\mathcal{A}_1$ outputs the message "Yes"; otherwise (i.e., either $\mathcal{A}$ does not produce a hypothesis or the produced hypothesis $h$ is not consistent with $\mathcal{E}$), $\mathcal{A}_1$ outputs the message "No".

Since $\mathcal{A}$ runs in polynomial time, $\mathcal{A}_1$ also runs in polynomial time. We have the following lemma which shows that $\mathcal{A}_1$ is indeed an **RP** algorithm for TA-FNF (proof in (Adiga et al., 2019)).

**Lemma 4.** *(i) If the* TA-FNF *instance has a solution, then Algorithm $\mathcal{A}_1$ produces the message "Yes" with probability at least* 0.9. *(ii) If* TA-FNF *instance does not have a solution, Algorithm $\mathcal{A}_1$ produces the message "No".*

Thus, starting with the assumption of an efficient learning algorithm $\mathcal{A}$, Lemma 4 shows that the TA-FNF problem is in **RP**, contradicting the assumption that **NP** $\neq$ **RP**. This completes our proof of Theorem 4. □

We note that Theorem 4 holds for the case of proper learning where the hypothesis class and the concept class coincide; here, they are both assumed to be $n$-tuples of threshold values. It is of interest to investigate whether the result can be extended to PAC learning under the representation-independent setting (see, e.g., (Warmuth, 1989)).

**Learning from positive examples.** Here, we show that there is an efficient consistent learner when all the examples in $\mathcal{E}$ are positive. The consistent learner is described in Algorithm 1. This is similar to an algorithm in (Adiga et al., 2017) for learning thresholds from observations. The idea behind the algorithm is the following. Suppose $(C_1,\ C_2) \in \mathcal{E}$. Consider any node $v$. Recall that score$(C_1, v)$ is the number of $1's$ provided by $C_1$ to the local function at $v$. Let $C_2(v)$ denote the state of $v$ in $C_2$. If $C_2(v) = 0$, then we can conclude that $t(v) >$ score$(C_1, v)$; otherwise, $t(v) \leq$ score$(C_1, v)$. For each node $v$, these inequalities can be used to find non-negative integers $\ell(v)$ and $u(v)$ such that $\ell(v) \leq t(v) \leq u(v)$. It is easy to see that there is a consistent assignment of threshold values to nodes only if for each node $v$, the range $R(v) = [\ell(v) \,..\, u(v)]$ is non-empty. Since there are at most $|\mathcal{E}|$ inequalities for each node $v$, the time needed to construct the range of threshold values for each node and check the feasibility is $O(|\mathcal{E}||V|)$, which is a polynomial function of the input size. Therefore, we have the following result.

**Algorithm 1** A consistent learner for positive examples.

```
Require: The underlying graph $G(V, E)$ of a SyDS and a collection $\mathcal{E}$ of positive examples.
Ensure: If a solution exists, then a consistent assignment of threshold values to the nodes in $V$.
 1: $X = \emptyset$.
 2: for all $(C_1,\ C_2) \in \mathcal{E}$ do
 3:     for all $v \in V$ do
 4:         if $C_2(v) = 0$ then
 5:             Add the inequality "$t(v) >$ score$(C_1, v)$" to $X$.
 6:         else
 7:             Add the inequality "$t(v) <$ score$(C_1, v)$" to $X$.
 8:         end if
 9:     end for
10: end for
11: If there is a solution to the set of inequalities in $X$, output a solution. Otherwise, output "no consistent assignment".
```

**Theorem 5.** *When all the examples are positive, there is an efficient algorithm to determine whether there is a consistent threshold assignment; if so, the algorithm finds one such assignment.* □

**Learning with a small number of negative examples.** Theorem 4 points out the difficulty of developing an efficient consistent learner for threshold SyDSs when there are both positive and negative examples. Since there is an efficient learner for the situation when the input has only positive examples (Theorem 5), it is seen that the computational intractability result arises due to the negative examples. Our next result shows that when the number of negative examples is *small* (i.e., $O(\log n)$ where $n$ is the number of nodes), an efficient consistent learner can be developed.

Let $\mathcal{E}_N$ be the set of negative examples. Given a negative example $x = (C_1,\ C_2) \in \mathcal{E}_N$, we say that a pair $(v, t)$ "handles" $x$ if setting the threshold of $v$ to $t$ makes the value of $v$ in the successor of $C_1$ to be *different from* $C_2(v)$. There might be multiple pairs $(v, t), (v', t')$, which handle an example $x \in \mathcal{E}_N$. Note that $v$ may or may not be same as $v'$, and $t$ may or may not be the same as $t'$. Also, $t$ should be within the range $R(v)$ determined from examining the positive examples $\mathcal{E}_P$. Let $S(v, t) = \{x \in \mathcal{E}_N : (v, t) \text{ handles } x\}$. Let $B = \{(v, t) : S(v, t) \neq \emptyset\}$. We have the following lemma (proof in (Adiga et al., 2019)).

**Lemma 5.** *There exists a consistent threshold assignment for $\mathcal{E}$ if and only if there exists a subset $B' = \{(v_1, t_1), \ldots, (v_r, t_r)\} \subseteq B$ where $v_1, \ldots, v_r$ are distinct nodes, and $\bigcup_{i=1}^{r} S(v_i, t_i) = \mathcal{E}_N$, such that threshold values $t_i$ for node $v_i$ are consistent with the ranges $R(v)$ inferred from the positive examples $\mathcal{E}_P$.*

We develop a dynamic programming algorithm to check whether there is a consistent learner when there are positive and negative examples. Let $v_1, \ldots, v_n$ be an ordering of the nodes. The algorithm uses the following steps.

(1) We maintain information in a table $M[j, S]$ where $j = 1, \ldots, n$ and $S$ ranges over all the subsets of $\mathcal{E}_N$. We define $M[j, S] = 1$ if there exists an assignment $\{(v_1, t_1), \ldots, (v_{j'}, t_{j'})\}$ for $j' \leq j$, such that all $x \in S$ are handled by this assignment; otherwise $M[j, S] = 0$. The entries of $M[\cdot, \cdot]$ are computed as follows.

(2) The base case is $j = 1$. For each $(v_1, t) \in B$, we set $M[1, S'] = 1$ for all $S' \subseteq S(v_1, t)$. We set $M[1, S] = 0$ for all remaining $S$.

(3) We set $M[j+1, S] = 1$ if $M[j, S] = 1$ or if $M[j, S - S(v_{j+1}, t)] = 1$ for some $(v_{j+1}, t) \in B$; otherwise, we set $M[j+1, S] = 0$.

The lemma below (proved in (Adiga et al., 2019)) establishes the correctness and the running time of the algorithm.

**Lemma 6.** *The above dynamic program has space and time complexity $O(2^{|\mathcal{E}_N|} poly(n))$, which is polynomial if $|\mathcal{E}_N| = O(\log n)$. Further, $M[n, \mathcal{E}_N] = 1$ if and only if there exists a consistent threshold labeling for $\mathcal{E}$.*

**Approximately satisfying constraints in $\mathcal{E}_N$.** We use the notation and a result from (Călinescu et al., 2011). A partition matroid $\mathcal{M} = (X, \mathcal{I})$ is defined in the following manner: $X$ is partitioned into $\ell$ sets $X_1, \ldots, X_\ell$, with associated integers $k_1, \ldots, k_\ell$. A set $A \subseteq X$ is independent iff $|A \cap X_i| \leq k_i$ for all $i$. A function $f : 2^X \to \mathbb{R}_+$ is submodular if for all $P \subseteq Q \subset X$, $p \notin Q$, we have $f(P \cup \{p\}) - f(P) \geq f(Q \cup \{p\}) - f(Q)$. Finally, $f$

is monotone if $f(P) \leq f(Q)$ for all $P \subseteq Q$. Călinescu et al. (2011) design an algorithm that gives a $(1 - 1/e)$-approximation for a monotone submodular function with matroid constraints.

Using our earlier notation, define $B(v) = \{(v,t) \in B : S(v,t) \neq \emptyset\}$ and $X = B$. Define the function $g : 2^X \rightarrow \mathbb{R}_+$ as $g(P) = |\bigcup_{(v,t)\in P} S(v,t)|$. Our algorithm involves the following steps.

(1) $B(v_1), \ldots, B(v_n)$ is a partition of $X$. We fix $k_i = 1$ for $i = 1, \ldots, n$. The matroid $\mathcal{M} = (X, \mathcal{I})$ is as above.

(2) Use the algorithm of (Călinescu et al., 2011) to find a subset $P \in \mathcal{I}$ that has the maximum value $g(P)$.

(3) The set $P = \{(v_{j_1}, t_{j_1}), \ldots, (v_{j_r}, t_{j_r})\}$ gives the threshold assignment.

The following lemma gives the performance guarantee provided by the above algorithm.

**Lemma 7.** *The assignment returned by the above algorithm ensures that at least* $(1 - 1/e)$ *fraction of the examples in* $\mathcal{E}_N$ *are handled.*

The lemma follows by noting that function $g(\cdot)$ is monotone submodular and then applying Theorem 1.1 from (Călinescu et al., 2011).

## 5. Experimental Results

**Analysis procedures and networks.** Figure 3 provides an overview of the major steps in the experiments. Mined graphs are obtained from the web and synthetic networks of different classes are generated; see Table 1. There are five graph instances for each of the random regular (RR) and scale-free (SF) synthetic networks for a specified $n$ (number of nodes). For each graph instance, five sets of true threshold assignments are made, where the threshold for each node $v$ is chosen uniformly at random from the range $[0, d_v+2]$ (the two limiting cases: 0 means the node will always change to state 1; $d_v + 2$ means the node will not transition to state 1 and will transition to state 0 if it is 1). For each true assignment of thresholds to nodes, Algorithm 1 is used to estimate ten inferred threshold assignments. This is done because the outcome of Algorithm 1 is stochastic due to the fact that the positive examples are chosen randomly. These sets of threshold estimates are also generated for different numbers $m$ of queries (or configurations), ranging from 10 to $10^5$. Each true threshold assignment, represented by $\mathcal{F}$, is compared to each of the estimated threshold sets (for each particular value of $m$), represented by $\mathcal{F}'$, by counting the fraction $f_{ne}$ of times that $\mathcal{F}(C) \neq \mathcal{F}'(C)$. Based on preliminary analyses, we take the number $n_t$ of configurations $C$ (i.e., comparisons or trials) to be $n_t = 10n$. In Algorithm 1 for estimating thresholds and in the evaluation of the dynamics between $\mathcal{F}$ and $\mathcal{F}'$, configurations are re-

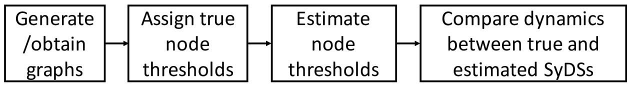

Generate /obtain graphs
Assign true node thresholds
Estimate node thresholds
Compare dynamics between true and estimated SyDSs

*Figure 3.* Overview of pipeline work items for experiments. The third box uses Algorithm 1.

quired. For consistency, configurations must be sampled from the same distribution for both operations. We evaluate two distributions. One configuration distribution $\mathcal{D}_u$ is a uniform distribution where each node is set to state 1 with probability $p$ (and to state 0 with probability $(1-p)$). This distribution has been analyzed in the context of learning halfspaces (Long, 1995). The second configuration distribution $\mathcal{D}_{PL}$ generates a probability in (0,1) uniformly at random, and using a power law relationship, determines the number $n_1$ of nodes whose state will be 1 in the configuration (He et al., 2016). Then, uniformly at random $n_1$ nodes are selected from $V$ to be in state 1, with the remaining nodes in state 0. Since our experiments use positive examples only and the underlying system is deterministic, once a random configuration $C$ is chosen, its successor $C'$ is determined. Thus, in our experiments, we don't sample pairs of configurations. We perform the steps of Figure 3 using a full factorial design, and results are presented next.

*Table 1.* Mined and synthetic networks, and their attributes.

| Network | Properties | | | |
|---|---|---|---|---|
| | $n$ | $\|E\|$ | $d_{ave}$ | $d_{max}$ |
| Jazz | 198 | 2742 | 27.70 | 100 |
| NRV | 769 | 4551 | 11.84 | 20 |
| euEmall | 986 | 16064 | 32.58 | 345 |
| Ran Reg$^{a,1}$ | 11–1000 | $n\ d_{avg}/2$ | 10 | 10 |
| Scl free$^{a,2}$ | 20–1000 | $\sim n\ d_{avg}/2$ | 9.5–9.9 | 13–149 |
| Cliques$^{3}$ | 400 | $n_q n_c(n_c-1)/2$ | $n_c - 1$ | $n_c - 1$ |

[a] 5 replicates per $n$ value.
[1] $n$ values are 11, 20, 40, 60, 80, 100, 200, 400, 600, 800, 1000.
[2] $n$ values are 20, 40, 60, 80, 100, 200, 400, 600, 800, 1000.
[3] number $n_c$ of nodes per clique are 10, 20, 40, 80, 100, 200, 400.

We use the above methods to investigate the behavior of the PAC model for different $(i)$ network classes, sizes of networks, $d_{ave}$, and graph density; $(ii)$ threshold estimation parameters such as configuration distributions $\mathcal{D}$, node probability $p$, and numbers $m$ of queries for Algorithm 1; and $(iii)$ $(\epsilon, \delta)$ pairs of the PAC model. Selected results are provided here; others are in (Adiga et al., 2019).

**Effects of graph size and number of queries.** Figure 4 shows $f_{ne}$ vs. $m$ for the mined networks. The first plot shows boxplots for the Jazz network. Each box represents 50 values of $f_{ne}$. (This is because there are 5 threshold assignments, and for each assignment, we generated 10 estimated threshold assignments.) As expected, as the number $m$ of queries increases, the fraction of configurations $C$ for which $\mathcal{F}(C) \neq \mathcal{F}'(C)$ decreases. In the second plot, the average values of $f_{ne}$ are plotted against $m$ for the three mined networks. The curves, highest to lowest, correspond to decreasing numbers of nodes in the graphs. That is, the larger the graph in terms of numbers of nodes, all other things being equal, the larger is the number of queries required to get accurate thresholds from Algorithm 1 because distance-1 neighbors in a graph typically do not make progress in narrowing threshold estimates in one step.

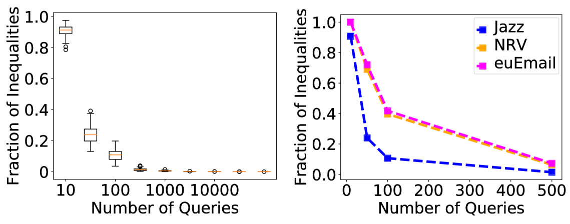


*Figure 4.* Fraction of inequalities $f_{ne}$ versus $m$ for mined networks (a) Jazz and (b) for the three mined networks. In (a), the number of queries ranges from 10 to 100,000. As $m$ increases from 10 to 100,000, $f_{ne} \to 0$. In (b) the focus is smaller values of $m$; errors are lesser in the Jazz network and greatest in euEmail as these networks have the minimum and maximum numbers of nodes of the mined graphs. There are 50 data points per $m$ value on the x-axis (1 graph instance $\times$ 5 true threshold assignments per graph $\times$ 10 estimated instances per true instance). In both plots, we use $\mathcal{D}_u$ with $p = 0.25$.

**Effect of graph density on $f_{ne}$.** For cliques (densest graphs) and independent sets of nodes (least dense graphs), theoretical results show that $f_{ne}$ is the same in both extreme cases. Hence, a natural question is what happens at intermediate values of graph density. To study this, we take a series of RR graphs and SF graphs with $d_{ave} = k = 10$ and increase $n$ from 11 (a clique) to 1000 (a less dense graph) to determine the fraction of errors in comparing $\mathcal{F}$ and $\mathcal{F}'$. Figure 5 shows $f_{ne}$ vs. $n$ for all of the random regular (RR) and scale-free (SF) networks. In this plot "$n$ increasing" is synonymous with "graph density decreasing." Each box represents 250 values of $f_{ne}$. As expected, in the first plot (RR) median $f_{ne}$ values are comparable at extreme ends. However, at intermediate values of $n$ (and density), the behavior is nonlinear and non-monotonic, showing increased $f_{ne}$. In the second plot, the effect exists in that data are shown for the smallest $n = 20$. For $n = 11$, the SF network would become a clique (keeping $d_{ave}$ the same) and the data from the plot at $n = 11$ shows smaller $f_{ne}$ values. The two different graph structures suggests that this effect may be robust across different network structures.

**Effect of distributions for sampling configurations.** We investigate the effect of different distributions—$\mathcal{D}_u$ and $\mathcal{D}_{PL}$—for sampling configurations for generating estimated thresholds and for comparing $\mathcal{F}$ and $\mathcal{F}'$. Results are shown in Figure 6 for RR networks. For $\mathcal{D}_u$ in the first plot, we take $p = 0.25$, and comparatively small $f_{ne}$ are observed.

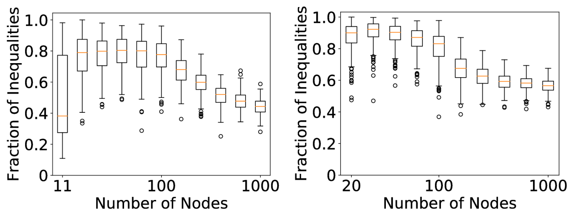


*Figure 5.* Fraction of inequalities $f_{ne}$ as a function of $n$ (and as a function of graph density, because graph density decreases as $n$ increases, for fixed value of $d_{ave}$) for (a) random regular (RR) and (b) scale-free (SF) networks. In both plots, $q = 0.1$, so the number of queries scales with $n$ according to $m = q\ n$. There are 250 data points per box plot. These plots illustrate that $f_{ne}$ increases at intermediate values of graph density, and decreases with very high and very low density. We use $\mathcal{D}_u$ with $p = 0.25$.

In contrast, for $\mathcal{D}_{PL}$ in the second plot, $f_{ne} = 1$ for almost all $n$. A fundamental difference between the two models is that for $\mathcal{D}_u$, the expected number $n_1$ of nodes in state 1 in each sampled configuration is the same ($n_1 = pn$), while for $\mathcal{D}_{PL}$, $n_1$ can vary as $0 \leq n_1 \leq n$. As a result, there may be less consistency in the configurations produced by $\mathcal{D}_{PL}$, leading to greater inconsistency in configurations for estimating thresholds and for comparing $\mathcal{F}$ and $\mathcal{F}'$.

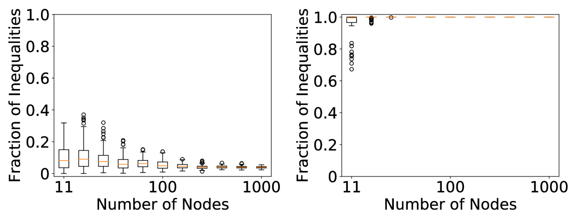


*Figure 6.* Fraction of inequalities $f_{ne}$ as a function of $n$ for different distributions of configurations: (a) $\mathcal{D}_u$ with $p = 0.25$ and (b) $\mathcal{D}_{PL}$. These results, on RR networks, show that widely different $f_{ne}$ can result from different models for generating configurations. Here, $m = qn$ with $q = 0.1$.

## 6. Summary and Future Work

We examined the learnability of local functions of SyDSs under the PAC model. Our focus was on threshold functions with emphasis on sample complexity, efficiency of learning and effect of network structure. There are several possible directions for future work. For example, one may consider learning other classes of local functions including stochastic functions. More general observation models can be explored. For example, observations can span multiple time steps. The difficulty here is that a single error in one step can lead to errors in the ensuing steps. Another and challenging research direction is to investigate learning both the graph topology (which we currently assume is known) and the local functions under the PAC model.

## Acknowledgments

We thank the reviewers for providing valuable feedback. This work is supported by DTRA CNIMS (Contract HDTRA1-11-D-0016-0001), NSF DIBBS Grant ACI-1443054, NSF EAGER Grant CMMI-1745207, NSF BIG DATA Grant IIS-1633028 and NSF CRISP 2.0 Grant 1832587. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

## References

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. V., Ravi, S. S., and Vullikanti, A. K. PAC learnability of node functions in networked dynamical systems. Technical Report, Network Systems Science and Advanced Computing Division, Biocomplexity Institute and Initiative, University of Virginia, Charlottesville, VA, 2019.

Amini, H. and Fountoulakis, N. Bootstrap percolation in power-law random graphs. *Journal of Statistical Physics*, 155(1):72–92, 2014.

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

Bei, X., Chen, W., Garg, J., Hoefer, M., and Sun, X. Learning market parameters using aggregate demand queries. In *Proc. AAAI*, pp. 411–417, 2016.

Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. K. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM (JACM)*, 36(4):929–965, 1989.

Brugere, I., Gallagher, B., and Berger-Wolf, T. Y. Network structure inference, a survey: Motivations, methods, and applications. *ACM Computing Surveys (CSUR)*, 51(2), 2018. (24 pages).

Călinescu, G., Chekuri, C., Pál, M., and Vondrák, J. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011.

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

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

Guille, A., Hacid, H., Favre, C., and Zighed, D. A. Information diffusion in online social networks: A survey. *ACM SIGMOD Record*, 42(2):17–28, 2013.

Hanneke, S. The Optimal Sample Complexity of PAC Learning. *J. Machine Learning Research*, 17:1–15, 2016. ISSN 15337928. URL http://arxiv.org/abs/1507.00473.

Haussler, D. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial intelligence*, 36(2):177–221, 1988.

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.

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.

Kuhlman, C. J., Kumar, V. A., Marathe, M. V., Swarup, S., Tuli, G., Ravi, S., and Rosenkrantz, D. J. Inhibiting the diffusion of contagions in bi-threshold systems: Analytical and experimental results. In *AAAI Fall Symposium: Complex Adaptive Systems*, 2011.

Lokhov, A. Reconstructing parameters of spreading models from partial observations. In *Advances in Neural Information Processing Systems*, pp. 3467–3475, 2016.

Long, P. M. On the sample complexity of PAC learning half-spaces against the uniform distribution. *IEEE Transactions on Neural Networks*, 6(6):1556–1559, 1995.

Miller, J. C. Spread of infectious disease through clustered populations. *Journal of the Royal Society Interface*, 6 (41):1121–1134, 2009.

Mitchell, T. M. *Machine Learning*. McGraw Hill, Boston, MA, 1997.

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

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

Newman, M. E. Spread of epidemic disease on networks. *Physical review E*, 66(1):016128:1–016128:12, 2002.

Romero, D. M., 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.

Schelling, T. C. *Micromotives and Macrobehavior*. Norton, New York, NY, 1978.

Valente, T. W. Social network thresholds in the diffusion of innovations. *Social Networks*, 18:69–89, 1996.

Valiant, L. G. A theory of the learnable. *Communications of the ACM*, 18(11):1134–1142, 1984.

Warmuth, M. K. Towards representation independence in PAC learning. In *Proc. International Workshop on Analogical and Inductive Inference (AII'89)*, pp. 78–103, Oct. 1989.

Watts, D. J. A simple model of global cascades on random networks. *Proceedings of the National Academy of Sciences*, 99:5766–5771, 2002.