ALGORITHMS ON GRAPH-STRUCTURED DATA WITH IMPERFECT INFORMATION

BY

HUOZHI ZHOU

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

Adviser:

Assistant Professor Lav R. Varshney

ABSTRACT

Graph-structured data is able to characterize pairwise or even higher-order relations among different data points, and has been demonstrated to be highly advantageous in various data mining and machine learning applications. Such graph-structured data may either come from real life networks, or some transformation based on data points. However, in practice the measurement of graph-structured data is usually partially incomplete or incorrect. For example, the measured states of nodes in the graph might be incorrect due to sensor noise. In this thesis, we study two problems on graph-structured data with imperfect information: hypergraph-based active learning and source estimation on directed acyclic graphs (DAGs), all with provable statistical guarantees.

In the first part of this thesis, we propose an active learning scheme which is able to accommodate the structure of hypergraphs, termed HS^2. HS^2 generalizes the previously proposed S^2 algorithm which is only able to solve graph-based active learning (GAL) with pointwise oracle. Our HS^2 is more flexible in the sense that it is adaptable for three different types of oracles: pointwise oracle, pairwise oracle, as well as noisy pairwise oracle. Based on a novel parametric system particularly designed for hypergraphs, we derive theoretical results on the query complexity of HS^2 for the above described settings. Both the theoretical and empirical results show that HS^2 outperforms the naïve combination of clique expansion and GAL algorithms.

Next we develop a heuristic, termed generalized Jordan center (GJC), to estimate the source of a spreading process on a DAG based on noisy and incomplete observations. This problem is motivated by contamination diffusion in a food supply chain. For this setting, identifying the source correctly and efficiently as well as inferring states of unobserved events are of top priorities (the *recall* problem). We believe this is the first work on source estimation with noisy information. Under mild conditions, GJC is the

maximum likelihood (ML) estimator of the diffusion source. Our proposed heuristic is parameter-free (only needs to know the structure of the DAG and states of some nodes), and can be evaluated efficiently by a message-passing-like algorithm in $\tilde{O}(|V|)$ complexity (the tilde notation means ignoring the logarithm factor), where V is the vertex set. Experiments on both synthetic and real networks show that GJC has significant gains over a naïve extension of Jordan center and is comparable to the exact ML estimate, in terms of source detection probability and false negative rate for recall.

To my parents, for their love and support.

ACKNOWLEDGMENTS

My thanks go to my great adviser, Prof. Lav R. Varshney, for his constructive suggestions, inspiring discussions, and endless encouragement throughout the two years of my master's study. Meanwhile, he also gives much freedom to explore different fields of research to learn a breadth of useful fundamental knowledge. He is not only my academic advisor, but also a role model in life philosophy. Initially I always doubted myself when doing research, but I have come gradually to enjoy it. I look forward to working as a PhD student in the next few years with him on the topic of interplay between safety and reinforcement learning.

I would like to thank Eli Chien and Pan Li for the collaboration on the hypergraph-based active learning, as well as my research mentor Ashish Jagmohan from IBM for the collaboration on the source estimation problem with partial and noisy observations. I am also very grateful to my labmates: Jianhao Peng, Chuanyi Zhang, Srilakshmi Pattabiraman, Ravi Kiran Raman, Daewon Seo and Xiou Ge for the many useful discussions we have. I have been extremely grateful to be a part of this community.

Finally, I owe special thanks to my parents for their selfless support throughout my life. Without their love, support, and encouragement, I would not be the person I am.

This work was funded in part by the IBM-Illinois Center for Cognitive Computing Systems Research (C3SR), a research collaboration as part of the IBM AI Horizons Network.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Graph-Structured Data

Machine learning on graphs is an important and ubiquitous task with applications ranging from recommendation to community detection. A graph is defined by a collection of nodes and edges, which may either come from real life networks such as social networks [1], or some transformation based on data points, e.g. nearest neighbor graphs [2]. The structure of a graph itself is able to characterize useful information for solving many engineering problems, and principally exploiting the structure of graphs has shown outstanding performance in applications from different fields. In other words, graphs are emerging as a powerful analysis paradigm for numerous problems. Much data can be modeled as a graph, and following is a short list of concrete examples:

- Cities are nodes and highways are edges (in a transportation network)
- Humans are nodes and relationships between them are edges (in a social network)
- Atoms are nodes and chemical bonds are edges (in a molecule)

Graph-structured data is becoming prevalent in many data mining and machine learning applications, but there are still many challenges with regard to this type of data. One main challenge is the scale of the data, especially in the era of big data. It is typical that social networks nowadays have billions of nodes; thus, developing efficient algorithms for solving problems on graph-structured data is extremely important. Another challenge in processing graph data is the error in the measurement step, which is always inevitable. For example, due to the RFID sensor noise, the state of nodes in a supply

chain network might be incorrect with some probability [3, 4]. Therefore the robustness of algorithms designed for graph data is also of great concern. In this thesis, we target two problems on graph data and develop efficient and robust algorithms for them respectively, with provable statistical guarantees.

1.2 Two Inference Problems on Graphs

Various inference problems can emerge in the context of graph data, and they can be divided into two categories: online problems and offline problems. The first type refers to situations where the program is operating and taking in new information in real time. The second type is opposite to the first type, in the sense that we have a static set of input data. One issue largely ignored by the research community is the unseen error in the graph data. This thesis studies the following two questions considering the unseen error in the graph data, which represent the above two types, respectively.

Active Learning on Hypergraphs This problem refers to the situation where the learner aims to select a minimum number of most informative nodes in the hypergraph, in order to correctly predict the labels of remaining unknown nodes with high confidence. Active learning on graphs has been studied extensively; most of the proposed algorithms for this problem fall into two categories: algebraic [5–7] or topological [8]. However, work on active learning for hypergraphs is quiet limited. In this thesis, we show the need for algorithms which can accommodate the structure of hypergraphs, since the projection from hypergraph to graph leads to loss of information.

Source Estimation with Noisy and Incomplete Observations Diffusion processes in complex networks characterize numerous real-world phenomena including disease spread in epidemics, rumor propagation in societies, and perhaps less well-known: contamination diffusion in food supply chains [9, 10]. Since such infections cause tremendous societal losses [11, 12], it is important to *identify* the source accurately and efficiently, so proper control strategies can inhibit or even eliminate spreading. Previous works on source estimation mostly focus on graphs with perfect information [13, 14], i.e., the states of all nodes and graph structure are known to be correct. However, in practice this assumption does not hold, due to the noise introduced in the graph data generation. We want to study how to infer the diffusion

source given noisy and incomplete observation of node states. In this thesis, we make some progress on solving this problem.

Bibliographical Note

The problem of active learning on hypergraphs and part of the results of Chapter 2 have been presented at the conference listed below and appear in its proceedings:

- I. Chein, H. Zhou, and P. Li, "HS^2: Active learning over hypergraphs with pointwise and pairwise queries," *in Proceedings of 22nd International Conference on Artificial Intelligence and Statistics (AISTATS)*, Apr. 2019.

The problem of source estimation with incomplete and noisy observations and part of the results in Chapter 3 have been presented at the conference listed below and appear in its proceedings:

- H. Zhou, A. Jagmohan, and L. R. Varshney, "Generalized Jordan center: A source localization heuristic for noisy and incomplete observations," *in Proceedings of the IEEE Data Science Workshop (DSW)*, June 2019.

CHAPTER 2

ACTIVE LEARNING ON HYPERGRPAHS

In this chapter, we study the problem of active learning on hypergraphs. We first state the concise problem formulation, and then demonstrate the active learning algorithms as well as their query complexity bounds, designed for different types of oracles respectively. In the last part, we show the experimental results of our proposed algorithms on both synthetic and real data.

2.1 Problem Formulation

We use $G = (V, E)$ to denote a hypergraph with a node set V and a hyperedge set E. A hyperedge $e \in E$ is a set of nodes $e \subset V$ such that $|e| \geq 2$. When for all $e \in E$, $|e| = 2$, G reduces to a graph.

Suppose that each node belongs to one of k classes. Let $[k]$ denote the set $\{1, 2, ..., k\}$. A *labeling function* is a function $f : V \mapsto [k]$ such that $f(v)$ is the label of node v. Given the labels of all nodes, we call a hyperedge e a *cut hyperedge* if there exist $u, v \in e, f(u) \neq f(v)$. The *cut set* C includes all cut hyperedges. Moreover, define the boundary of the cut set C as $\partial C = \bigcup_{e \in C} e$, i.e., the set of nodes that appear in some cut hyperedges. By removing all the cut hyperedges, we suppose that G is partitioned into T connected components whose node sets are denoted by $V_1, V_2, ..., V_T$. For any pair of connected components V_r, V_s, define the associated *cut component* as $C_{rs} = \{e \in C : e \cap V_r \neq \emptyset, e \cap V_s \neq \emptyset\}$. Note that two different cut components of hyperedges C_{rs} and $C_{r's'}$ may have intersection in the hypergraph setting and the union of C_{rs} for all (r, s) pairs is the cut set C.

We are considering active learning problems, in which the learner is allowed to ask queries and collect information from the oracle. In this work, we study two kinds of oracles: the pointwise oracle $\mathcal{F}_0 : V \mapsto [k]$ and the pairwise oracle

$\mathcal{O}_0 : V \times V \mapsto \{0, 1\}$, which are defined as follows. For all $v_1, v_2 \in V$,

$$\mathcal{F}_0(v_1) = f(v_1),\ \mathcal{O}_0(v_1, v_2) = \begin{cases} 1, & \text{if } f(v_1) = f(v_2) \\ 0, & \text{if } f(v_1) \neq f(v_2). \end{cases}$$

In the pairwise setting, we also allow for a noisy oracle, denoted by $\mathcal{O}_p$, where p stands for the error probability of the oracle, i.e.,

$$P(\mathcal{O}_p(v_1, v_2) = \mathcal{O}_0(v_1, v_2)) = 1 - p.$$

We assume that for different pairs of nodes, the responses of the oracle are mutually independent. However, for each pair of nodes (v_1, v_2), $\mathcal{O}_p(v_1, v_2)$ is consistently 0 or 1. Therefore, querying one pair multiple times does not lead to different responses or affect the learning performance.

We use the term *query complexity*, denoted by $\mathcal{Q}$, to quantify how many times an algorithm uses the oracle. Our goal is to design algorithms which learn the partition $V = \bigcup_{i=1}^{T} V_i$, or equivalently cut set C, with query complexity $\mathcal{Q}$ as low as possible. In this work, due to the randomness of the proposed algorithms, we focus on learning the exact C with high probability. That is, given a $\delta \in (0, 1)$, with probability $1 - \delta$ we recover C with query complexity $\mathcal{Q}(\delta)$.

Remark 2.1.1. The original S^2 paper considers a simpler noise model [8], where one allows independent responses after querying for a single event multiple times. In this case, a simple majority voting can be used for aggregating and denoising the information. However, according to real-life experiments in crowdsourcing [15, 16], such a method intrinsically introduces bias and thus majority voting may even increase the error. Therefore, we consider a more realistic model used in [15]. Also note that this noise model is not applicable to the case of pointwise oracle as the noise may always lead to some incorrect labels that can never be fixed.

2.2 HS^2 with a Pointwise Oracle

In this section, we propose the HS^2 algorithm with a pointwise oracle, termed HS^2-point, which essentially generalizes the S^2 algorithm for GAL [8] to the hypergraph setting. HS^2-point is similar to S^2, insofar as the algorithm only asks for the label of the midpoint of current shortest path among all paths that connect two nodes with different labels, while the path now is defined over hypergraphs. The novelty of HS^2-point appears in the corresponding analysis of the query complexity. We find that how well cut components are clustered determines the query complexity. Later, we will formally define it as the *clusteredness* of cut components. In contrast to [8], for HS^2-point, clusteredness of cut components is determined by a much more complicated measure that characterizes the distance between cut hyperedges. Moreover, we tighten the original analysis for S^2. As a corollary, the tightened bound shows that HS^2-point requires lower query complexity than a naive combination of the clique-expansion method and S^2.

We start by introducing the HS^2-point algorithm. As HS^2 depends on shortest paths, we first define a path in hypergraphs and its length.

Definition 2.2.1 (Path in hypergraph)**.** Given a hypergraph $G = (V, E)$, we say there is a path of length l between nodes $u, v \in V$ if and only if there exists a sequence of hyperedges $(e_1, e_2, ..., e_l) \subseteq E$ such that $u \in e_1$, $v \in e_l$ and $e_i \cap e_{i+1} \neq \emptyset \quad \forall i \in [l-1]$.

Conceptually, the algorithm operates by alternating two phases: random sampling and aggressive search. Each outer loop corresponds to a random sampling phase, where the algorithm will query randomly. This phase will end when two nodes with different labels are detected and there is a path connecting them, which is determined by the subroutine $MSSP(G, L)$ (it finds the midpoint on the shortest among all the shortest-paths that connect oppositely labeled vertices in the collection of labeled nodes). Then, the algorithm turns into the inner loop, i.e., the aggressive search phase that searches cut hyperedges. In the inner loop, cut hyperedges are gradually removed and G breaks into a collection of connected components. Here, L is a list to collect labeled nodes with labels. Algorithm 1 will keep tracking the size of L. When the query complexity budget is used up, the algorithm ends and outputs the remaining connected components of G.

Algorithm 1: HS^2-point

Input : A hypergraph G, query complexity budget $\mathcal{Q}(\delta)$
Output: A partition of V
Main Algorithm: $L \leftarrow \emptyset$
while *1* **do**
  $x \leftarrow$ Uniformly at random pick an unlabeled node.
  do
    Add $(x, \mathcal{F}_0(x))$ to L
    Remove all hyperedges containing nodes with different label from G.
    if *more than* $\mathcal{Q}(\delta)$ *queries are used* **then**
      Return the remaining connected components of G
    end
  while $x \leftarrow MSSP(G, L)$ *exists*
end

The aggressive search phase that finds all cut hyperedges within low query complexity is the most important step. The key idea is the following. On the path between two nodes with different labels, there must be at least one cut hyperedge. Intuitively, to efficiently find this cut hyperedge, we may use a binary-search method along the shortest one of such paths. That is, we iteratively query for the label of the node that bisects this path. The binary search and the search of a shortest path are done simultaneously by the key subroutine $MSSP(G, L)$ (Algorithm 2). Finding the shortest path in the hypergraph can be implemented via standard breath-first-search algorithm [17]. A more efficient way to search the shortest path in a dynamic hypergraph is described in [18]. Since we focus on query complexity, discussion of the time complexity of the algorithms is outside the scope of the paper.

To characterize the query complexity of Algorithm 1 we need to introduce the following concept.

Definition 2.2.2 (Balancedness)**.** We say that G is β-balanced if $\beta = \min_{i \in [k]} \frac{|V_i|}{n}$.

Definition 2.2.3 (Distance between cut hyperedges)**.** Let $d_{sp}^{G-C}(v, u)$ denote the shortest path between nodes v, u with respect to the hypergraph G after all cut hyperedges are removed. Let $\Omega_i(e) = \{x \in e | x \in V_i\}$. Define the directed distance between cut hyperedges as $\Delta : C \times C \to \mathbb{N} \cup \{0, \infty\}$: for

Algorithm 2: MSSP

Input : The hypergraph graph G, label list L
Output: The midpoint of shortest-shortest path
Main Algorithm:
for *each* $v, u \in L$ *such that* u, v *has different label* **do**
  $P_{v,u} \leftarrow$ shortest path between v, u in G.
  $l_{u,v} \leftarrow$ length of $P_{u,v}$.($= \infty$ if doesn't exist)
end
$(v^*, u^*) = \arg\min l_{u,v}$
if (v^*, u^*) *exists and* $l_{v^*,u^*} \geq 2$ **then**
  Return the midpoint of P_{v^*,u^*}.
else
  Return $\emptyset$
end

$e_1,\ e_2 \in C$,

$$\begin{aligned}
&\Delta(e_1, e_2) \\
&= \sup_{(i,j):e_1,e_2\in C_{i,j}} \Big(\sup_{v_1\in\Omega_i(e_1)} \inf_{u_1\in\Omega_i(e_2)} d_{sp}^{G-C}(v_1, u_1) \\
&+ \sup_{v_2\in\Omega_j(e_1)} \inf_{u_2\in\Omega_j(e_2)} d_{sp}^{G-C}(v_2, u_2) + 1 \Big). \qquad (2.1)
\end{aligned}$$

If e_1, e_2 do not belong to a common cut component, let $\Delta(e_1, e_2) = \infty$.

For e_1, e_2 that belong to certain cut component, the metric $\Delta(e_1, e_2)$ characterizes the length of shortest path that contains e_2 after we have found and removed e_1. With the above distance, we may characterize the clusteredness of cut hyperedges. First, we need to construct a dual directed graph $H_r = (C, \mathcal{E})$ according to the following rule: the nodes of H_r correspond to cut hyperedges of G and for any two nodes e, e', ee' is an arc in H_r if and only if $\Delta(e, e') \leq r$. According to the definition, each cut component $C_{i,j}$ is mapped to a group of nodes in H_r. Now, we may define κ-clusteredness.

Definition 2.2.4 (κ-clusteredness)**.** A cut set C is said to be κ-clustered if for each cut component $C_{i,j}$, the corresponding nodes in H_κ are strongly connected.

The intuition behind the above definition is that ideally we want the cut hyperedges in one cut component to not be so far away from another cut

hyperedge. For better understanding, suppose HS^2-point has found and removed the cut hyperedge e_1. Another hyperedge e_2 in the same cut component appears in the shortest path whose endpoints are in e_1. This parameter guarantees that HS^2-point needs only at most $\lceil \log_2 \kappa \rceil$ queries along such a path to find the cut hyperedge e_2. Hence, if the hypergraph has a small κ, we can efficiently find all the cut hyperedges in C after we find the first one in each cut component in the random sampling phase. Typically κ is not large, as κ is naturally upper bounded by the diameter of the hypergraph, which, in a small-world situation, is at most $O(\log n)$ [19].

The novel part of HS^2-point is that we propose Definition 2.2.3 and Definition 2.2.4, which properly generalize the parametric system of S^2 [8] to hypergraphs and leads to the following theoretical estimation of query complexity.

Theorem 2.2.5. Suppose that $G = (V, E)$ is β-balanced. The cut set C induced from a label function f is κ-clustered and m non-empty cut components. Then for any $\delta > 0$, Algorithm 1 will recover C exactly with probability at least $1 - \delta$ if $\mathcal{Q}(\delta)$ is larger than

$$\begin{aligned}\mathcal{Q}^*(\delta) \triangleq & \frac{\log(1/(\beta\delta))}{\log(1/(1-\beta))} + m(\lceil \log_2(n) - \log_2(\kappa) \rceil) \\ & + \min(|\partial C|, |C|)(\lceil \log_2(\kappa) \rceil + 1). \end{aligned} \tag{2.2}$$

Note that Theorem 2.2.5 not only generalizes Theorem 3 from [8] to the hypergraph case but also provides a tighter result. Specifically, it improves the original term $|\partial C|$ to $\min(|\partial C|, |C|)$. Recall the definitions of $|\partial C|$ and $|C|$. Typically, $|C| < |\partial C|$ corresponds to the case when the number of cut hyperedges is small while the size of each cut hyperedge is large, which may appear in applications that favor large hyperedges [20–22]. This improvement is also critical for showing that the HS^2-point algorithm has lower query complexity than a simple combination of CE and the original S^2 algorithm [8]. We will illustrate this point in the next subsection.

2.2.1 Comparison with clique expansion

Clique expansion (CE) is a frequently used tool for learning tasks over hypergraphs [23–26]. CE refers to the procedure that transforms hypergraphs into

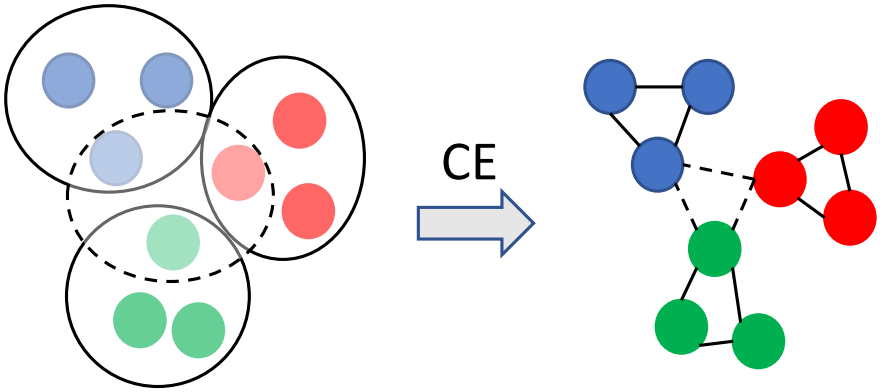

Figure 2.1: An example of clique expansion. Left: the orginal hypergraph G with 4 hyperedges; Right: the clique-expanded graph $G^{(ce)}$. The colors of nodes identify the labels and the dashed hyperedges/edges are cut hyperedges/edges.

graphs by expanding hyperedges into cliques. Based on the graph obtained via CE, one may leverage the corresponding graph-based solvers to solve learning tasks over hypergraphs. For HAL, we may choose a similar strategy. Suppose the obtained graph after CE is denoted by $G^{(ce)} = (V^{(ce)}, E^{(ce)})$, so that $V^{(ce)} = V$, and for u, $v \in V^{(ce)}$, $uv \in E^{(ce)}$ if and only if $\exists e \in E$ such that $u, v \in e$. In this subsection we will compare the bounds of query complexity of HS^2-point evaluated over G and that of S^2 evaluated over $G^{(ce)}$.

Suppose G is β-balanced, with m cut components, and the corresponding cut set C is κ-clustered. In the following proposition, we show that some parameters of $G^{(ce)}$ are the same as those of G.

Proposition 2.2.6. $G^{(ce)}$ is β-balanced and has exactly m cut components. Let $C^{(ce)}$ be the cut set of $G^{(ce)}$. Then, $C^{(ce)}$ is κ-clustered and $|\partial C| = |\partial C^{(ce)}|$. However, we always have $\min(|C|, |\partial C|) \leq \min(|C^{(ce)}|, |\partial C^{(ce)}|)$.

As graphs are special case of hypergraphs, Theorem 2.2.5 can be used to characterize the query complexity of S^2 over $G^{(ce)}$. For this purpose, recall the parameters in Theorem 2.2.5 that determine the query complexity. Combining them with Proposition 2.2.6, it is clear that the HS^2 algorithm often allows for lower query complexity than that of CE plus S^2 and such gain comes from the case when $|C| \leq |C^{(ce)}|$. To see the benefit of HS^2-point more clearly, consider the example in Figure 2.1. Once HS^2-point finds and removes the cut hyperedge of G, the correct partition of V is learned. So we only need to collect the labels of any two nodes in $|\partial C|$. However, if we use

S^2 over the obtained graph $G^{(ce)}$, all three nodes in $\partial C^{(ce)}(= \partial C)$ must be queried for labels before we learn the correct partition.

Remark 2.2.1. The benefit of HS^2-point essentially comes from the fact that $|C|$ is often smaller than $|C^{(ce)}|$. Note that the query complexity for S^2 derived in [8] does not reflect such a parametric dependence.

Remark 2.2.2. Note that in the example in Figure 2.1 we have $|C| \leq |C^{(ce)}|$ and $|\partial C| = |\partial C^{(ce)}|$. However, $|C|$ is not necessarily smaller than $|C^{(ce)}|$. Consider the following example: Suppose all nodes of G have different labels and there are $\binom{n}{3}$ hyperedges in E that cover all triples. Then, $G^{(ce)}$ is a big clique connecting all nodes. In this case $|C| = \binom{n}{3} > \binom{n}{2} = |C^{(ce)}|$. Nevertheless, in this case we have $|\partial C| = |\partial C^{(ce)}| < |C^{(ce)}|$ and hence Proposition 2.2.6 still holds. This example shows that it is non-trivial to prove Proposition 2.2.6.

2.3 HS^2 with Pairwise Oracle

We now look into the HAL problem with a pairwise oracle. Since the proposed algorithms also depends on the strategy of searching for the shortest path that connects two nodes with different labels, we refer to them as HS^2-pair. As mentioned, to our best knowledge, HS^2-pair appears to be the first model-free strategy to handle the HAL/GAL problems with a pairwise oracle.

We analyze settings with both noiseless and noisy oracles. The noiseless case is simple and will be introduced first. Then, we introduce the noisy case that is much more involved. Note that in the setting with a pairwise oracle, the exact label of each node is not known and not relevant. Hence, without loss of generality, we associate the ith class identified during the learning procedure with the label i.

2.3.1 Noiseless case

We start by introducing the setting with a noiseless pairwise oracle. The key point is to first seed a few classes and then classify a newly selected node via pairwise comparison with the seeds. If there is a match, we assign the node to its corresponding class; otherwise, we assign the node to a new class.

Notationally, we let $S_i, i \in [k]$ be the set of nodes that have been classified to the ith class so far. Each S_i starts from one node when a node from the ith new class is detected and S_i gradually grows when new nodes of this class are detected. As all nodes $u \in S_i$ share the same label, for a new node v, we use $\mathcal{O}_0(v, S_i)$ to denote the query $\mathcal{O}_0(v, u)$, $u \in S_i$. The HS^2-pair algorithm for the noiseless case is listed in Algorithm 3.

Algorithm 3: The noiseless HS^2-pair

Input : A hypergraph G and query complexity budget $\mathcal{Q}(\delta)$.
Output: A partition of V.
Main Algorithm: $L \leftarrow \emptyset$, #c $\leftarrow 1$
$v \leftarrow$ Uniformly at random pick an unlabeled node
Add (v, 1) to L and set $S_1 \leftarrow \{x\}$
while *1* **do**
  $v \leftarrow$ Uniformly at random pick an unlabeled node
  do
    Collect $\mathcal{O}_0(v, S_i)$ for all $i \in$ [#c]
    if $\exists i,\ \mathcal{O}_0(v, S_i) = 1$ **then**
      Add (v, i) to L and v to S_i
    else
      #c $\leftarrow$ #c + 1
      Add (v, #c) to L and Set $S_{\#c} \leftarrow \{v\}$
    end
    Remove all hyperedges containing nodes with different labels from G
    if *more than* $\mathcal{Q}(\delta)$ *queries are used* **then**
      Return the remaining connected components of G
    end
  while $x \leftarrow MSSP(G, L)$ *exists*
end

The only difference between HS^2-pair in the noiseless case and HS^2-point is the way to label a newly selected node. We leverage the pairwise oracle to compare the new node with each class that has been identified. Intuitively, we need at most k pairwise queries to identify the label of each node. Moreover, without additional assumptions on the data, it appears impossible to identify the label of each node with $o(k)$ many pairwise queries. Therefore, combining this observation with Theorem 2.2.5, we establish the query complexity of Algorithm 3 in the following corollary, which essentially is $\Theta(k)$ times the number of queries required by HS^2-point.

Corollary 2.3.1. Suppose $G = (V, E)$ is β-balanced. The cut set C is κ-clustered and the number of non-empty cut components is m. Then for any $\delta > 0$, Algorithm 3 will recover C exactly with probability at least $1 - \delta$ if $\mathcal{Q}(\delta)$ is larger than $kQ^*(\delta)$, i.e.,

$$k\frac{\log(1/(\beta\delta))}{\log(1/(1-\beta))} + km(\lceil \log_2(n) - \log_2(\kappa) \rceil) \\ + k\min(|C|, |\partial C|)(\lceil \log_2(\kappa) \rceil + 1).$$

2.3.2 Noisy case

We consider next the setting with a noisy pairwise oracle. The key idea is similar to the one used in the noiseless case: we first identify seed nodes for the different classes. Due to the noise, however, we need to identify a sufficiently large number of nodes within each class during Phase 1 so that the classification procedure in Phase 2 has high confidence. To achieve this, we adopt a strategy similar to that used in Algorithm 2 of [15] in Phase 1, which can correctly classify a group of vertices into different clusters with high probability based on pairwise queries as long as the size of each cluster is not too small. Phase 2 reduces to classifying the remaining nodes. In contrast to the noiseless case, we adopt a normalized majority voting strategy: we will compare the ratios of the nodes over different classes that claim to have the same label with the incoming node. We list our HS^2-pair with noise in Algorithm 4.

We now describe the query complexity of Algorithm 4.

Theorem 2.3.2. Suppose $G = (V, E)$ is β-balanced. The cut set C induced from a label function f is κ-clustered and has m non-empty cut components. Then for any $\delta > 0, p < \frac{1}{2}$, Algorithm 4 will recover C exactly with probability at least $1 - \delta$ if $\mathcal{Q}(\delta)$ is larger than

$$\mathcal{Q}^*(\delta/4)M + \frac{128Mk^2 \log M}{(2p-1)^4}, \tag{2.3}$$

Algorithm 4: HS^2-pair with noise

Input : A hypergraph G, query complexity budget $\mathcal{Q}(\delta)$, parameter M

Output: A partition of V

Phase 1:

Uniformly at random sample M nodes from G;

Use Algorithm 2 in [15] on these M nodes to get a partition $S_1, ..., S_k$.

Let $S = \bigcup_{i=1}^{k} S_i$;

Phase 2:

$L \leftarrow \{(v, i) | v \in S_i, i \in [k]\}$;

Remove all hyperedges whose containing different labels from G;

while *1* **do**

  Uniformly at random sample an unlabeled node v;

  do

    $M_i \leftarrow |\{u \in S_i | \mathcal{O}_p(u, v) = 1\}|$ for all $i \in [k]$;

    $i^* \leftarrow \arg\max_{i \in [k]} M_i / |\hat{S}_i|$, add (v, i^*) to L;

    Remove all hyperedges that contain different labels from G;

    if *more than* $\mathcal{Q}(\delta)$ *queries are used* **then**

      Return the remaining connected components of G

    end

  while $x \leftarrow MSSP(G, L)$ *exists*;

end

where $\mathcal{Q}^*(\delta)$ is defined in (2.2), and M is an integer satisfying

$$\begin{aligned} &\frac{M}{\log M} \geq \frac{128k}{\beta(2p-1)^4},\ M \geq \frac{12}{\beta}\log\frac{4k}{\delta}, \\ &M \geq \frac{8}{\delta},\ M \geq \frac{2}{\beta D(0.5||p)}\log\frac{8(k-1)\mathcal{Q}^*(\delta/4)}{\delta}. \end{aligned} \tag{2.4}$$

Here $D(p||q)$ denotes the KL-divergence of two Bernoulli distributions with parameters p and q.

We only provide a sketch of the proof of Theorem 2.3.2. For the complete proof, please refer to the appendix.

Proof. (sketch) In Phase 2, we expect to select $\mathcal{Q}^*(\delta_1)$ nodes for labeling, according to Theorem 2.2.5. This phase may require $M\mathcal{Q}^*(\delta_1)$ queries. To classify all these nodes correctly via normalized majority voting with probability at least $1-\delta_2$, we require each S_i to be large enough. Specifically, via the Chernoff bound and the union bound, we require

$$\min_{i\in[k]}|S_i| \geq \frac{1}{D(0.5||p)}\log\frac{2(k-1)\mathcal{Q}^*(\delta_1)}{\delta_2}. \tag{2.5}$$

To obtain a sufficiently large $|S_i|$, we need to sample a sufficiently large number of points M in Phase 1. With probability $1-k\exp(-M\beta/8)$, we can ensure that

$$\min_{i\in[k]}|S_i| \geq \frac{\beta M}{2}. \tag{2.6}$$

Combining (2.5) and (2.6) gives the fourth inequality in (2.4). Moreover, we also need to cluster these S_i correctly via Algorithm 2 in [15], which requires the first three constrains in (2.4) and the additional $\frac{128Mk^2\log M}{(2p-1)^4}$ queries according to Theorem 3 in [15]. This gives the formulas in Theorem 2.3.2. □

Remark 2.3.1. Suppose that the parameters (p, k, δ, β) are constants. Then, the fourth requirement of M in (2.4) reduces to $M = O(\log(\mathcal{Q}^*(\delta)))$, and the overall query complexity equals $O(\mathcal{Q}^*(\delta)\log(\mathcal{Q}^*(\delta)))$. Comparing this to Theorem 2.2.5 and Corollary 2.3.1, we only need $O(\log(\mathcal{Q}^*(\delta)))$ times more queries for the setting with the noisy pairwise oracle.

Recall the perfect partitioning according to the labels follows $V = \bigcup_{i=1}^{T} V_i$. If we additionally assume that T equals the number of classes k, Phase 1 of

Algorithm 4 will guarantee to sample at least one node from each $V_i, i \in [T]$. This observation allows us to get rid of the random sampling procedure in Phase 2. So the first term in $\mathcal{Q}^*(\delta)$ essentially vanishes. We may achieve the following tighter result.

Corollary 2.3.3. Suppose $G = (V, E)$ is β-balanced. The cut set C induced from a label function f is κ-clustered and m non-empty cut components. Moreover, suppose $T = k$. Then, for any $\delta > 0, p < \frac{1}{2}$, Algorithm 4 will recover C exactly with probability at least $1 - \delta$ if $\mathcal{Q}(\delta)$ is larger than

$$\mathcal{Q}_1^* M + \frac{128Mk^2 \log M}{(2p-1)^4}$$

where

$$\begin{aligned}\mathcal{Q}_1^* = & \, m(\lceil \log_2(n) - \log_2(\kappa) \rceil) \\ & + \min(|\partial C|, |C|)(\lceil \log_2(\kappa) \rceil + 1),\end{aligned}$$

and now M is the smallest integer satisfying

$$\begin{aligned}\frac{M}{\log M} &\geq \frac{128k}{\beta(2p-1)^4}, \\ ; M &\geq \frac{12}{\beta} \log \frac{3k}{\delta}, \\ M &\geq \frac{6}{\delta}, \\ ; M &\geq \frac{2}{\beta D(0.5||p)} \log \frac{6(k-1)\mathcal{Q}_1^*}{\delta}.\end{aligned}$$

In the end of this section, we remark on the CE method in the setting with the pairwise oracle. One still may first apply CE to obtain a graph $G^{(ce)}$ and then run Algorithms 3 and 4 over $G^{(ce)}$. Corollary 2.3.1 and Theorem 2.3.2 again indicate that the query complexity depends on $\min\{|C|, |\partial C|\}$. By using Proposition 2.2.6, we can again demonstrate the superiority of our proposed approaches over CE-based methods.

2.4 Experiments

In this section, we evaluate the proposed HS^2-based algorithms on both synthetic data and real data. We mostly focus on demonstrating the benefit of

HS^2 in handling the high-order structures. For the setting with a pointwise oracle, we compare HS^2-point with some GAL algorithms including the original S^2 [8] and EBM [6], a greedy GAL algorithm based on error bound minimization. To make these GAL algorithms applicable to our high-order data, we will first transform hypergraphs into standard graphs by clique expansion which was introduced in Section 2.2.1. For the setting with a pairwise oracle, as there are no other model-free algorithms even for GAL to the best of our knowledge, we compare HS^2-pair over hypergraphs with the combination of clique expansion and HS^2-pair over graphs (termed CE + S^2-pair later). All the results are averaged over 100 independent tests.

For the real datasets, we test both HS^2 and CE+S^2 on the task of motion segmentation, which is essentially a subspace clustering problem and typically needs to utilize hypergraph structures [27]. We use the popular benchmark — the Hopkins 155 — dataset [28] to evaluate the performance. As mentioned in [22, 29], the trajectories on the distinct motions can be grouped into 4-dimensional subspaces. We generate 5-uniform hypergraphs from the data, since a fit to d-dimensional subspace can only be evaluated over at least $d + 1$ data points. It is crucial to use hypergraphs instead of standard graphs for this tasks.

2.4.1 Synthetic data

For the synthetic data, we investigate the effects of the scale of hypergraphs n and the number of classes k on all proposed algorithms. We generate labeled hypergraphs according to the following random hypergraph model: fix the number of inner-cluster and intra-cluster hyperedges, and then generate all hyperedges uniformly at random without replacement. Specifically, we fix the size of all hyperedges to be 5, and restrict each cluster to be equal-sized. In our experiments, we uniformly randomly place $\frac{n}{k}\log\frac{n}{k}$ hyperedges within each cluster. This ensures that each cluster will be connected with high probability. Then we uniformly randomly place $\frac{1}{3}\frac{n}{k}\log\frac{n}{k}$ hyperedges across different clusters, which means $|C|=\frac{1}{3}\frac{n}{k}\log\frac{n}{k}$. For the experiments on pointwise queries, the results are shown in Figure 2.2. As it shows, HS^2-point outperforms S^2 and EBM with nontrivial gains (roughly by a factor of 2). The reason why the query complexity scales linearly in n is due to our

experiment setting. We place $\frac{n}{k}\log\frac{n}{k}$ hyperedges within each cluster which makes the κ small. Hence the query complexity is dominated by the last term in (2.2). In addition, we also vary the size of hyperedges and the number of cut edges. The results are shown in Figure 2.3. We can see that HS^2-point will perform better when the size of hyperedges gets larger. Also we observe that when the number of cut edges gets larger, the CE+S^2 approach needs to query all n nodes while the HS^2 approach does not have to. All these results suggest that our HS^2 is better than the CE+S^2 approach. For the

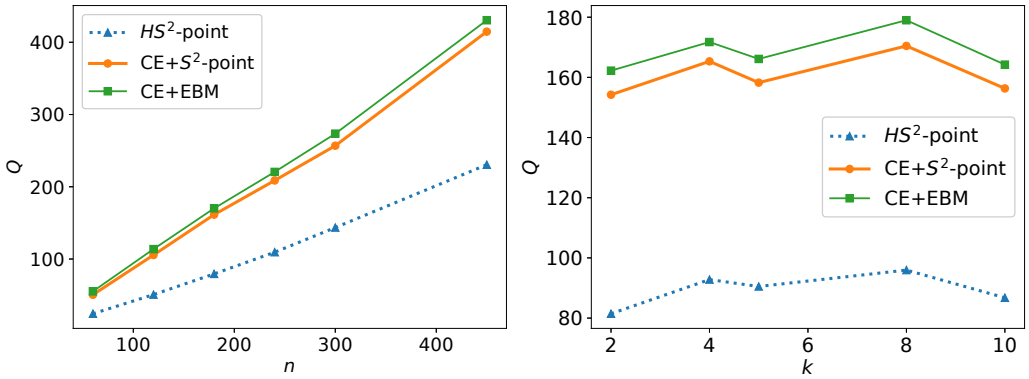

Figure 2.2: Simulation results with pointwise oracles over synthetic data. Left: query complexity vs scale of hypergraphs n with fixed $k = 3$; Right: query complexity vs the number of classes k with fixed $n = 200$.

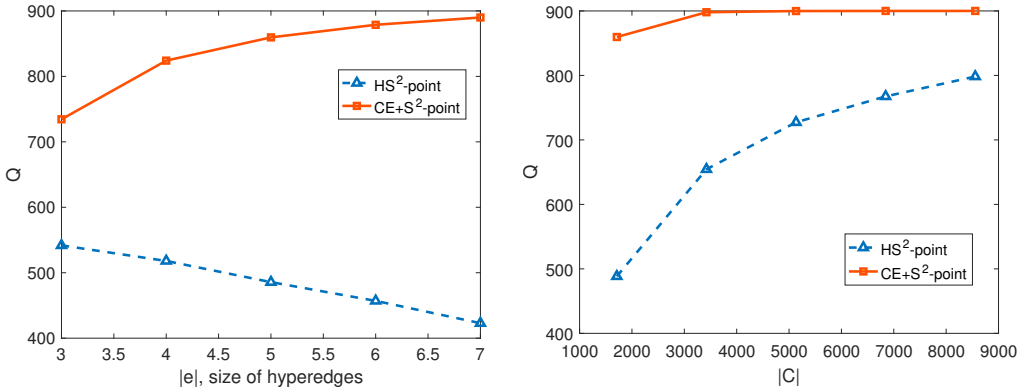

Figure 2.3: Simulation results with pointwise oracles over synthetic data. Left: query complexity vs size of hyperedges with fixed $k = 3$, $n = 900$; Right: query complexity vs the number cut edge $|C|$ with fixed $k = 3$, $n = 900$.

experiments on pairwise noiseless queries, the results are shown in Figure 2.4. Again, our HS^2-pair algorithm outperforms the naive combination of CE

and S^2-pair. In contrast to the pointwise case, we can see that the query complexity increases almost linearly with respect to the number of classes. This is because we need $\Theta(k)$ pairwise queries to identify the label of one node. To test HS^2-pair with noisy oracle, we construct a larger hypergraph,

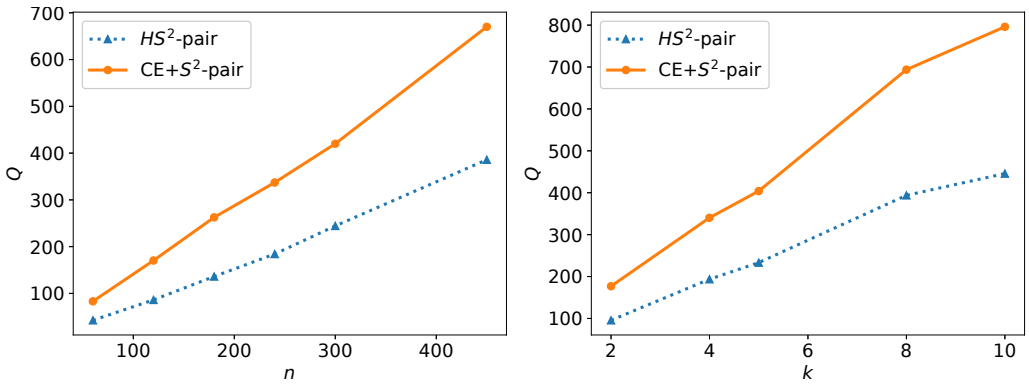

Figure 2.4: Simulation results with pairwise oracles over synthetic data. Left: query complexity vs scale of hypergraphs n with fixed $k = 3$; Right: query complexity vs the number of classes k with fixed $n = 200$.

due to the fact that in phase 1 it requires sufficient numbers of pairs of nodes to be queried. We set the total number of nodes in the hypergraph $n = 5000$, number of clusters $k = 2$ and the number of pairs of nodes to be queried in phase 1 $M = 2000$. The result is shown in Table 2.1. As expected, HS^2 is better than CE+S^2 in terms of query complexity.

Table 2.1: Query complexity with noisy pairwise oracle on synthetic graphs.

	noisy HS^2-pair	CE+noisy S^2-pair
Query Complexity	5,401,830	8,574,332

2.4.2 Real world application

We test the algorithms on 4 checkerboard sequences in the Hopkins 155 dataset under the pointwise query setting. They are sequences of indoor scenes taken with a handheld camera under controlled conditions. The checkerboard pattern on the objects is used to assure a large number of tracked points. We follow the same methodology as [22] to generate hypergraphs from these data. For each cluster i with n_i points, we sample $n_i \log n_i$ subsets of 5 points from them, as a $4D$-subspace was required to be fitted on each sample via SVD. We denote $N = \sum_i n_i$ to be the number of total

inner-cluster samples. We place a hyperedge on the sampled subset if the sum of the distance of corresponding points to the fitted $4D$-subspace is less than a threshold. Then we sample $\frac{N}{6}$ subsets of 5 points uniformly at random among all points and place hyperedges by following the same criterion. The results are in the Figure 2.5. We can see that indeed HS^2 needs many fewer queries than CE+S^2, which matches our theoretical and synthetic results.

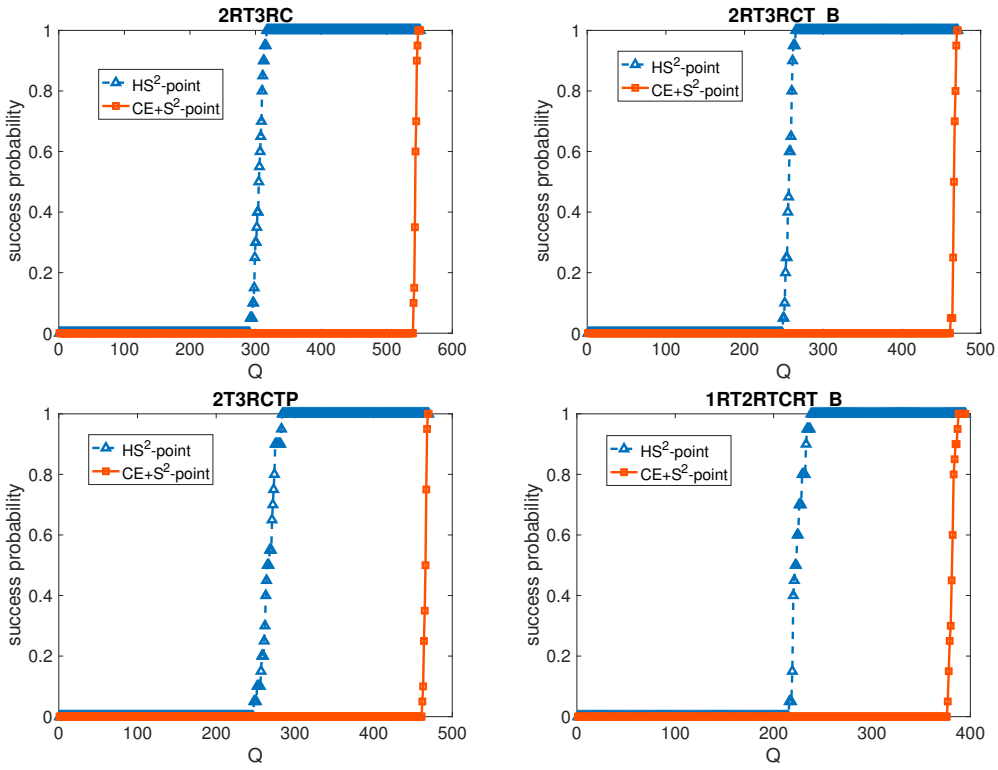

Figure 2.5: Results for the experiments on the Hopkins 155 dataset. The title for each subfigure describes how the corresponding data is obtained. For example, 2RT means that object 2 translates and rotates on the same axis. Each task contains about 400 to 500 points.

CHAPTER 3

SOURCE ESTIMATION WITH NOISY AND INCOMPLETE OBSERVATIONS

In this chapter, we will first state the problem formulation of source estimation with noisy and incomplete observations explicitly, and then introduce our proposed heuristic, generalized Jordan center (GJC) which extends the previous Jordan center [14], as well as its performance guarantee. Lastly, we will demonstrate the outstanding performance of our proposed heuristic on both synthetic and real data. According to our experiments, GJC is comparable to the exact maximum likelihood estimator, and significantly outperforms the naïve extension of previously proposed method source estimator for perfect information.

3.1 Problem Formulation

Consider the DAG $G = (V, E)$. At time $t = 0$, the infection process starts from source $s \in V$ following the transmission model defined in [30], having key properties:

- Infection process travels independently through graph,
- Infection of any susceptible node only depends on nodes that directly infect it, and
- Any infected node remains infected for all time thereafter.

The model is parameterized by a first-order, row-stochastic transition matrix P_T, where entry $P_T(i, j)$ is the probability node j is infected by node i in one time step. Assume we only have access to the state of a subset of nodes, through a binary symmetric channel with crossover probability η. That is, $\forall n \in O$, where O is the set of nodes observed as infected:

$$P(n \text{ is infected} | n \in O) = 1 - \eta$$
$$P(n \text{ is uninfected} | n \in O) = \eta$$

From the Bayesian perspective, we must find the most likely source $s^* \in C$, where C is the candidate set of sources, based on O. (In general, O may not be the collection of all infected nodes but just a small subset.) Given these definitions, the Bayes estimate of the diffusion source is:

$$s^* = \arg\max_{s \in C} P(s|O) = \frac{P(s)P(O|s)}{P(O)}, \tag{3.1}$$

where $P(s|O)$ is the conditional probability of node being source s given the infection pattern O. If we do not have prior information on the source, the MAP estimator reduces to the ML estimate, $s^* = \arg\max_{s \in C} P(O|s)$. The exact joint likelihood function for noiseless observations is [30]:

$$P(O|s) = \prod_{i=1}^{k} P(s \to o_i) = \prod_{i=1}^{k} \{(I - P_T)^{-1} P_T\}_{so_i}, \tag{3.2}$$

where $P(s \to o_k)$ is the total probability of reaching the kth infected node o_k from starting point s along all possible infection paths, and $\{(I-P_T)^{-1}P_T\}_{so_i}$ is the so_i-th entry of the matrix $(I - P_T)^{-1}P_T$. The expression is due to the independence assumption of the contamination process. Note that $(I-P_T)^{-1}$ is well-defined since for any absorbing Markov chain $I - P_T$ is invertible [31]. Thus the computation complexity of evaluating the ML estimate of the contamination source is $O(|V|^w)$, where $2 < w < 3$ is the best constant to implement matrix inversion and multiplication [32].

This strategy is computationally intractable when dealing with large-scale networks even for noiseless observations. For noisy observations, it becomes much worse since it additionally requires enumerating all possible state combinations, which in worst-case is $2^{|V|}$. The exact joint likelihood function of node s being the source given noisy observation O is:

$$P(O|s) = \frac{1}{Z} \sum_{i=1}^{m} \eta^{h(\hat{O}_i, O)} P(\hat{O}_i|s), \tag{3.3}$$

where $\frac{1}{Z}$ is the normalization constant, $\{\hat{O}_i\}_{i=1}^m$ is the collection of all feasible infection patterns (all infected nodes share at least one common ancestor)

and $h(\cdot,\cdot)$ is the Hamming distance between binary vectors. We slightly abuse notation with vectorized versions of O and $\hat{O}_i$ having length $|O|$ in $h(\cdot,\cdot)$, and for each entry, 1/0 represents infected/uninfected. Now we are ready to introduce the definition of GJC.

3.2 Generalized Jordan Center

Definition 3.2.1. The *generalized Jordan center* is:

$$\hat{s}_{GJ} = \arg\min_{s\in B} \max_{n\in\hat{O}(s)} d(s,n),$$

where $\hat{O}(s)$ is the subset of observed infected nodes O, such that any node in $O\backslash\hat{O}(s)$ is not connected to node s, and B is the collection of nodes such that $\forall\, n \in B$, $n = \arg\max_{s\in V} \sum_{m\in D(s)} \mathbb{1}\{m \in O\}$ where $D(s)$ is the collection of descendent nodes of s and s itself. We term B as the set of nodes with the highest beliefs.

Before introducing the performance guarantee of GJC, we need some technical definitions and lemmas.

Definition 3.2.2. A DAG is *layered* if its adjacency matrix can be written in the following form, after permuting node labels:

$$\mathbf{A} = \begin{bmatrix} \mathbf{0} & \mathbf{A}_{1,2} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{A}_{2,3} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & & & \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{A}_{k-1,k} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \end{bmatrix},$$

where $\mathbf{A}_{i,i+1}(i \in [k-1])$ is the adjacency matrix that defines the connectivity pattern from layer i to layer $i+1$, and k is the number of layers.

Roughly, for a given DAG, its vertex set can be partitioned into k disjoint subsets (layers) $\{L_1, \dots, L_k\}$, and nodes in L_{i+1} can only be connected from nodes in $L_i, i \in [k-1]$. Figure 3.1 shows examples of layered and non-layered DAGs to clarify the definition.

Lemma 3.2.3. For any layered directed graph G, if O is error-free, the most likely contamination source is one of the Jordan centers: $s^* \in \{\hat{s}_J\}$, where

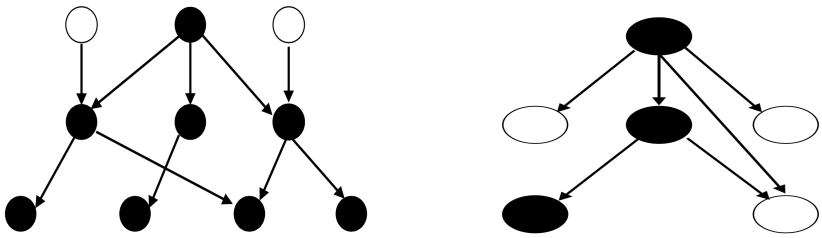

Figure 3.1: Examples of layered (left) and unlayered (right) DAGs. Node color indicates status: black means infected and white means uninfected. Clearly, these DAGs are feasible since all infected nodes share at least one common ancestor.

a Jordan center is $\hat{s}_J = \arg\min_{s \in C} \max_{n \in O} d(s,n)$, $d(s,n)$ is the shortest distance from node s to node n, and C is the candidate set of possible contamination sources. By definition, any node in the set C is connected to all of the infected nodes in O.

Proof. Let $l(n)$ be the layer of node n, $n \in L_{l(n)}$. Let s^* be the most likely contamination source in the $l(\hat{s}_J)$th layer, i.e. $P(O|s^*) \geq P(O|n), \forall\, n \in L_{l(\hat{s}_J)}$. Further, by definition of Jordan center, if $l(n) > l(\hat{s}_j)$, then $n \notin C$ (shown by contradiction). Thus we only need to analyze nodes from the first layer L_1 to $L_{l(\hat{s}_J)-1}$. Considering any node $n \in L_{l(\hat{s}_j)-1}$:

$$P(O|n) = \sum_{m \in L_{l(\hat{s}_j)}} P(n \to m) P(O|m) \leq P(O|s^*).$$

By induction, we know that $\forall\, n \in L_i, i \in \{1, ..., l_{\hat{s}_j}\}$, $P(O|s^*) \geq P(O|n)$. Thus the most likely contamination source must be one of the Jordan centers. □

Note that unlike undirected tree-like graphs, DAGs can have more than two Jordan centers [14]. Lemma 3.2.3 states that if our partial observations are error-free and the network structure is layered, we can narrow our search space for the most likely diffusion sources to the collection of Jordan centers, regardless of the underlying dynamics P.

More generally, the set of observed infected nodes can be infeasible, i.e. the observed infected nodes do not have any common ancestors. This is due to the causality of DAGs. Clearly for any connected undirected graph, any infection pattern is feasible. As we discuss before, computing the exact joint likelihood function is computationally intractable for large networks but GJC can be evaluated efficiently, which we will discuss later.

Let us first dig into the property of GJC a little bit. Clearly when O is error-free, GJC is equivalent to the standard Jordan center. We motivate the heuristic as follows.

Lemma 3.2.4. The $\hat{s}_{GJ}$ is the most likely diffusion source given the most likely infection pattern if the underlying network is a *layered* DAG, i.e. $\hat{s}_{GJ} = \arg\max_{n \in V} P(\hat{O}^*|n)$ where $\hat{O}^* = \arg\max_{i \in [m]} P(\hat{O}_i|O)$.

Proof. By the i.i.d. assumption of the observation noise,

$$P(\hat{O}_i|O) \propto \eta^{h(\hat{O}_i, O)}, \forall i \in \{1, 2, ..., m\}.$$

Then combining Lem. 3.2.3 and Def. 3.2.1 yields the result. □

Notice that to evaluate the GJC, for each node $n \in V$ we must compute the subset of observed infected nodes O which includes all descendants of this node in O (denoted B_n), as well as the maximum distance to its observed infected descendants d_n. These two quantities can be computed efficiently by the following local update rule. For any node $n \in V$, we have

$$B_n = \begin{cases} \cup_{m \in c(n)} B_m, & n \notin O \\ (\cup_{m \in c(n)} B_m) \cup n, & n \in O \end{cases} \tag{3.4}$$

$$d_n = \max_{m \in c(n)} d_m + 1, \tag{3.5}$$

where $c(n)$ is the collection of children nodes of n, i.e. there exists at least one path from n to any node in $c(n)$. Thus Algorithm 5 uses this update rule to efficiently find the GJC for arbitrary DAGs with complexity $O(|O||V|+|B|)$. But if $|O| \ll |V|$ and $|B| \ll |V|$ (reasonable since most likely observations are very limited), then the complexity is roughly $O(|V|)$. Now we are ready to state the performance guarantee of GJC. Under mild conditions, GJC is the ML estimate for layered DAGs.

Theorem 3.2.5. Let $G = (V, E)$ be an arbitrary layered DAG and η be the parameter of the observation noise. If $\hat{s}_{GJ}$ is unique, there exists a computable constant $c > 0$ such that if $\eta < c$ then $\hat{s}_{GJ}$ is the ML estimate.

Proof. Let $F = \{\hat{O}_1, \hat{O}_2, ..., \hat{O}_m\}$ be the collection of all possible subsets of O that are feasible, in decreasing order of probability ($\hat{s}_{GJ}$ is the Jordan

Algorithm 5: GJC Message Passing

Phase 1: Evaluate beliefs, maximum distance $\forall n \in V$
for $n \in V$ **do**
  $B_n \leftarrow \phi$;
  $d_n \leftarrow 0$;
  if $n \in O$ **then**
    $B_n \leftarrow B_n \cup n$;
  end
end
for $n \in O$ **do**
  Propagate ID, relative distance to ancestors $A(n)$;
  if $a \in A(n)$ **then**
    $B_a \leftarrow B_a \cup n$;
    $d_a \leftarrow \max(d_a, d(a, n))$
  end
end
Phase 2: Ranking
Select the collections of nodes with highest beliefs
$B = \arg\max_{n \in V} |B_n|$;
Evaluate GJC $\hat{s}_{GJ} = \arg\min_{n \in B} d_n$, ties break uniformly at random
return $\hat{s}_{GJ}$

center associated with $\hat{O}_1$). Denote the layer of $\hat{s}_{GJ}$ by L_m. By Lem. 3.2.4, $P(\hat{O}_1|\hat{s}_{GJ}) \geq P(\hat{O}_1|n), \forall n \in V$. The likelihood function of noisy observation O given the source to be $\hat{s}_{GJ}$ is:

$$P(O|\hat{s}_{GJ}) = \frac{1}{Z} \sum_{i=1}^{m} \eta^{h(\hat{O}_i, O)} P(\hat{O}_i|\hat{s}_{GJ}). \tag{3.6}$$

By the uniqueness assumption of $\hat{s}_{GJ}$, $\arg\max_{i \in m} |\hat{O}_i|$ is unique which is $\hat{O}_1$. Then any other elements in F are a subset of $\hat{O}_1$. Using an argument similar to the proof of Lemma 3.2.3, we can show that $\forall\, n \in A(\hat{s}_{GJ})$, $P(O|\hat{s}_{GJ}) \geq P(O|n)$, where $A(n)$ is the collection of ancestor nodes of n. In addition, $\forall\, n \in D(\hat{S}_{GJ})$, the difference of joint likelihood functions associated with $\hat{s}_{GJ}$ and n, is

$$\begin{aligned} & P(O|\hat{s}_{GJ}) - P(O|n) \\ & = \frac{1}{Z} \eta^{h(O, \hat{O}_1)} P(\hat{O}_1|\hat{s}_{GJ}) \\ & \quad - \frac{1}{Z} \sum_{i=2}^{m} \eta^{h(O, \hat{O}_i)} P(\hat{O}_i|n)(1 - \prod_{j=1}^{|\hat{O}_i|} P(\hat{s}_{GJ} \to n)). \end{aligned} \tag{3.7}$$

Dividing the expression by $\eta^{h(O,\hat{O}_1)}$, we have all $n \in D(\hat{s}_{GJ})$.

$$\frac{P(O|\hat{s}_{GJ}) - P(O|n)}{\eta^{h(O,\hat{O}_1)}} \geq \frac{1}{Z} P(\hat{O}_1|\hat{s}_{GJ}) - \frac{1}{Z}(m-1)\eta.$$

So if $\eta < \frac{P(\hat{O}_1|\hat{s}_{GJ})}{m-1}$, $\hat{s}_{GJ}$ is the ML estimate. Note that when we analyze the difference, w.l.o.g. we assume n is a candidate source for all infection patterns in F except $\hat{O}_1$. For other cases, the analysis still holds. □

The condition for $\hat{s}_{GJ}$ to be the exact ML estimate is intuitive for layered DAGs. If the noise is small enough, the most likely infection pattern dominates the joint likelihood function and therefore does not impact optimality. Theorem 3.2.5 further gives some insight into when this heuristic performs poorly. By analyzing (3.7), we see GJC performs poorly when the observed infected nodes are not evenly distributed over the network or the noise is high.

3.3 Recall

Another practical concern arises in food supply chain networks is the so called *recall* problem, which aims to correctly predict the states of remaining unknown nodes given the information we have gathered so far. Thus in addition to the source estimation probability, another reasonable yet important metric to measure the performance of the source estimate is the false negative rate. To be more specific: Given the contamination source estimate (evaluated by different methods) and the collection of node states which are observed, what is the prediction error on the unknown node states? In this thesis, we apply the maximum likelihood estimator to predict the unknown node states, assuming the parameters of the network spreading dynamic are known. We present the empirical performance of our proposed source estimate heuristic in the following section.

3.4 Experiments

Though our theoretical analysis focused on layered DAGs, we now see it performs well in experiments on general DAGs.

We evaluate GJC over synthetic and real networks, comparing to the naïve combination of Jordan center and random guess (JC+R), i.e., evaluate the Jordan center for feasible infection patterns and randomly guess otherwise, as well as the exact ML estimate. Source detection probability and false negative rate in recall are our performance metrics. The false negative rate is $|\hat{U} \cap I|/|V|$, where $\hat{U}$ is the set of nodes estimated to be uninfected and I is the ground truth set of infected nodes. This is important in food safety since infected nodes should not be misclassified. When evaluating false negative rate, we assume access to the network dynamics P_T, and infer the unknown node states by ML estimation given the source estimator $\hat{s}$ and noisy observation O. All simulation results are averages over 100 independent trials.

Random Networks We investigate effects of network scale and of noise on GJC and JC+R, generating simulated data as follows: (1) Generate a random layered DAG with adjacency matrix A by fixing the number of nodes (10) in each layer and connecting each pair of nodes between adjacent layers with probability 0.2. (2) Generate a $|V|\times|V|$ random matrix M where each entry is an i.i.d. uniform random variable. (3) Obtain transition matrix P_T by computing the Hadamard product $M \cdot A$ and then normalize each row to make it row-stochastic. Given structure A and dynamics P_T, we run Monte Carlo simulations to evaluate JC+R, GJC, and ML, by randomly choosing the diffusion source, and running spreading over 20 timesteps.

Figure 3.2 shows experimental results. GJC has non-trivial gains for source detection probability and false negative rate, compared to JC+R. As the fraction of observed nodes increases, the detection probability generally increases. On the other hand, increasing the noise makes identifying the correct diffusion source much harder. The gain of GJC comes from careful handling of infeasible infection patterns, by choosing the most likely source associated with the most likely infection pattern given the noisy observations. Surprisingly, the performance of GJC is comparable to exact ML estimation.

Synthetic Supply Chain All three proposed methods are evaluated on a synthetic supply chain network, which is generated by IBM's work on trusted supply chain using blockchain. The following is a description of the supply-chain process, and a contamination graph model which approximately models

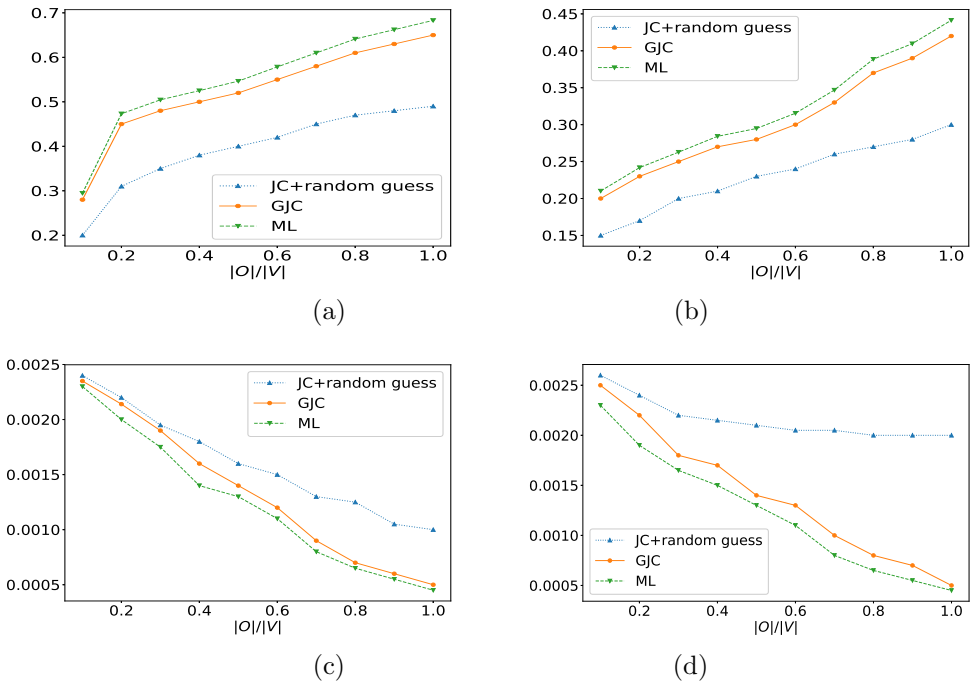

Figure 3.2: Random layered DAGs. Detection probability: (a) $\eta = 0.1$. (b) $\eta = 0.2$. False negative rate: (c) $\eta = 0.1$. (d) $\eta = 0.2$.

the reality.

- Step 1: Harvest a batch of raw fruit at a farm.
- Step 2: Transform the batch into packaged raw fruits cases.
- Step 3: Ship the cases to warehouses/manufacturing plants (MFs).
- Step 4: Receive the cases at the warehouse/MF.
- Step 5: Transform each raw fruit case into processed fruit cases.
- Step 6: Ship the processed fruit cases from warehouse/MF to distribution center (DC).
- Step 7: Receive the processed fruit cases at DC.
- Step 8: Transform - essentially mix the received cases into cases that will be sent to stores. (The simulator does not allow mixing, unfortunately, but see below for how our contamination graph model can model mixing.)

- Step 9: Ship from DC to store.
- Step 10: Receive at store.

The contamination graph model is generated according to the following rules, which are able to coarsely characterize:

- Each node represents an event which comes along with some side information. The side information includes location, date, step ID, and process ID.
- Connect an edge from event i to event j if the two events are part of the same process and event j is the consecutive event of i, i.e., step ID of j = step id of $i + 1$.
- In addition, connect an edge from event i to event j, if the two events
  - belong to different processes.
  - have the same step ID, which can only be 1, 2, 5 or 8.
  - happen at the same location, and the time difference is within a specified threshold.

We simulate 100 processes (each process has 10 event nodes) and then aggregate them according to rules mentioned before, to construct the contamination network. We further assume the probability of contamination spread along an edge should be 100 for shipping and receiving, and less than 100 for commission, transform and cross-process edges (in the experiment, it is uniform random variable ranges from 0 to 1). The experimental results, which are all averages of 100 independent tests, are summarized in Figure 3.3.

Google+ Dataset We also evaluate GJC on tame Google+ dataset with 107614 nodes and 13673453 edges [33], which is a general directed graph (not layered). In our simulation, we assume uniform diffusion rates for all out-edges of each node, i.e., $\forall\ (i,j) \in E, P_T(i,j) = \frac{1}{deg_+ i}$ (edges with the same source node have the same diffusion rate). We run the experiments by choosing the source with the greatest number of connected nodes, i.e. $\arg\max_{i \in V} |\{j : j \in V, d(i,j) < \infty\}|$ and running the spreading over 10 timesteps.

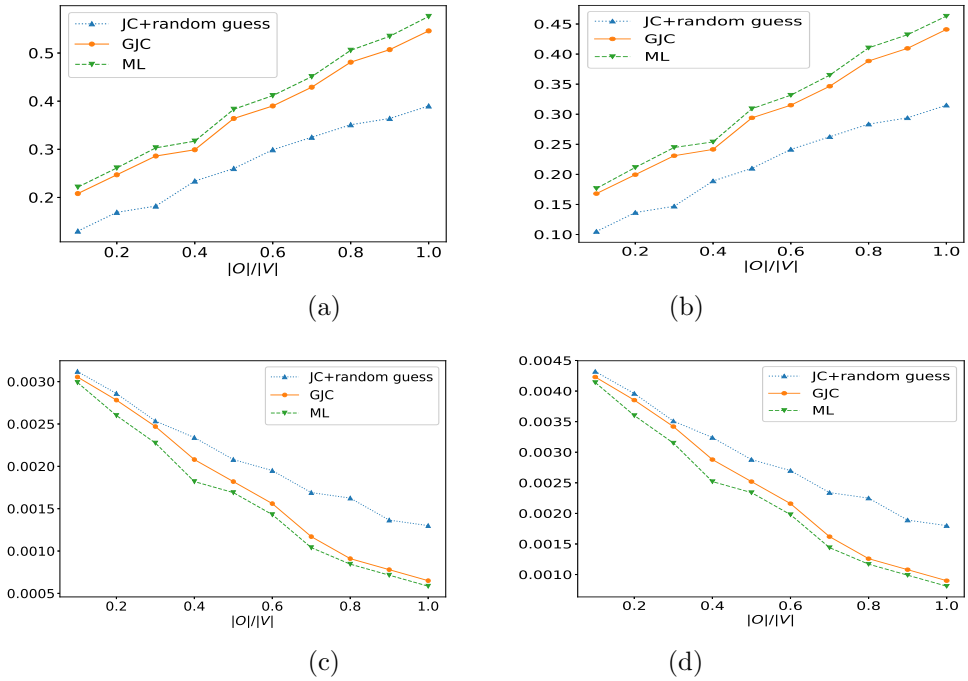

Figure 3.3: Synthetic food supply chain. Detection probability: (a) $\eta = 0.1$. (b) $\eta = 0.2$. False negative rate: (c) $\eta = 0.1$. (d) $\eta = 0.2$.

Experimental results in Figure 3.4 show GJC is comparable to the exact ML estimate. Results also indicate the performance gap between GJC and JC+R is much more for real networks, since the real network is sparser than the random layered DAGs we studied. Thus as the fraction of observed nodes increases, the observation is more likely to be infeasible. This explains why for JC+R, the detection probability decreases and false negative rate increases as the fraction of observed nodes increases.

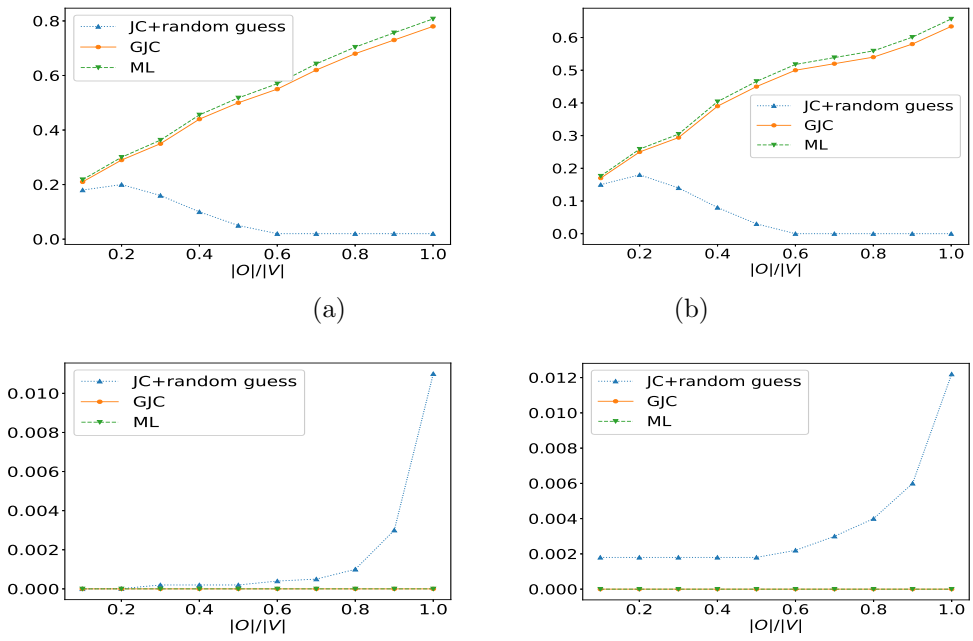

Figure 3.4: Google+. Detection probability: (a) $\eta = 0.1$. (b) $\eta = 0.2$. False negative rate: (c) $\eta = 0.1$. (d) $\eta = 0.2$.

CHAPTER 4

CONCLUSION

In this thesis, we study efficient algorithms which are robust to data noise, for two typical problems on graphs: selecting a minimum number of most informative nodes in a hypergraph in order to correctly predict the labels of remaining nodes, and estimating the diffusion source of a contamination process on DAG with noisy and incomplete observations.

For the active learning problem, we use the majority voting strategy to overcome the effect of noise. Compared to the case of noiseless oracles, the query complexity is penalized by a factor of $\log M$, where M is query complexity bound for the case of noiseless oracle. In the source estimation problem, our strategy is to rule out the infeasible infection pattern and evaluate the Jordan center with respect to the most likely infection pattern. This method works well in the low-noise regime, since the most likely infection pattern dominates the joint likelihood function when the noise is reasonably small.

Future work involves the interplay between these two problems, i.e., how to actively query the node states in order to locate the diffusion source. It is also interesting to derive the converse bound for the HAL problem; for now, we only have the achievability bound but we do not know whether our algorithm is order-optimal. Our guess is this algorithm is not order-optimal, since [8] constructs a hard example which shows that the original S^2 algorithm is near-order optimal for this instance. Even though our proposed algorithm is not order-optimal, we are still interested in what is the gap between the converse and achievability. Lastly, there are also lots of unexplored problems in source estimation under imperfect information. In this thesis, we only consider the case when the node state observations are noisy; another possibility is that the structure of the graph itself is uncertain. One other potential extension is how to generalize to the case of multiple diffusion sources, which is more practical but much more challenging.

APPENDIX A

DETAILED PROOFS FOR HS^2

A.1 Proof of Theorem 2.2.5

We need Lemma 1 of [8] which characterizes a bound for the query complexity of the random sampling phase. We first define a *witness set* of the cut set C as the node set that contains at least one node for each $V_i, i \in [T]$.

Lemma A.1.1 (Lemma 1 in [8])**.** Consider a β-balancedness graph $G = (V, E)$. For all $\alpha > 0$, a subset W chosen uniformly at random is a witness of the cut set C with probability at least $1 - \alpha$ as long as

$$|W| \geq \frac{\log\left(\frac{1}{\beta\alpha}\right)}{\log\left(1/(1-\beta)\right)}.$$

Moreover, we will need the following lemma. Basically it ensures that once HS^2-point discovers a cut hyperedge from a cut component, then HS^2-point will discover all remaining cut hyperedges in this cut component and the shortest paths that include these hyperedges are at most of length κ.

Lemma A.1.2. Suppose a hypergraph G with a cut set C is κ-clustered. Moreover, suppose C_{rs} is a cut component. If $e \in C_{rs}$ is discovered, which means a pair of nodes $u \in \Omega_r(e), v \in \Omega_s(e)$ are labeled, then at least one remaining cut hyperedge in C_{rs} lies in a path of length at most κ from a pair of nodes with labels r and s respectively.

Proof. By definition 2.2.4, we know that the hyperedges in C_{rs} will form a strongly connected component in H_κ. This means for any $e \in C_{rs}$, there is at least one $e' \in C_{rs}$ such that the arc ee' exists in H_κ. Recall there exists arc ee' in H_κ if and only if $\Delta(e, e') \leq \kappa$. By definition 2.2.3 this means for any node pair $u \in \Omega_r(e), v \in \Omega_s(e)$, the length of the shortest path including e' but excluding e will be less then κ. Note that in the definition of $\Delta(e, e')$, we

use the supremum taking over the node set $\Omega_r(e), \Omega_s(e)$. This is because it ensures that no matter which node pair $u, v \in e$ we have, $\Delta(e, e')$ can always upper bound the length of the shortest path including e' but excluding e with endpoints u, v. In contrast, we use the infimum taking over the node set $\Omega_r(e'), \Omega_s(e')$. This is because it only needs to search for the shortest path. Hence once we find a cut hyperedge e in C_{rs}, we are guaranteed to find at least one cut hyperedge $e' \in C_{rs}$ through a path of length $l_S \leq \kappa$ after we remove e. □

Now let us prove Theorem 2.2.5. The proof follows an outline similar to that of proof given in [8]. However, we need to take care of the hypergraph structures that are described by the Definitions 2.2.1 to 2.2.4. We will also derive a tighter bound for the number of runs R, which finally yields a lower query complexity than that in [8].

A.2 Proof of Proposition 2.2.6

A.2.1 Checking the equal parameters

We check the parameters one by one.

We start from proving that if C is κ-clustered, then $C^{(ce)}$ is also κ-clustered. We note that performing CE does not change the length of the shortest path of arbitrary node pair $v_1, v_2 \in V$. This is because CE will replace a hyperedge by a clique, which makes all nodes in the hyperedge become fully connected. Hence the $C^{(ce)}$ is still κ-clustered.

Now, we prove that if G has m non-empty cut components, then $G^{(ce)}$ will also have m non-empty cut components. We note that for any non-empty cut component $C_{i,j}$ in G, there is at least one hyperedge $e \in C_{i,j}$. By definition, we know that $e \cap V_i \neq \emptyset$ and $e \cap V_j \neq \emptyset$. So after CE, in the clique corresponding to this hyperedge e, there must be at least one edge such that one of its endpoints is from V_i and the other is from V_j, which makes $C_{i,j}$ still non-empty in $G^{(ce)}$. On the other hand, for arbitrary i, j, the cut component $C_{i,j}$ is empty in G if and only if there is no hyperedge between V_i, V_j. Hence $C_{i,j}$ will still be empty in $G^{(ce)}$. Together we show that if there are m non-empty cut components in G, there are exactly m non-empty cut components

in $G^{(ce)}$.

It is easy to see $G^{(ce)}$ keep β-balanced as f does not change in CE. Now, we prove that $|\partial C|= |\partial C^{(ce)}|$. For any $e \in C$, denote $e = \{v_1, ..., v_d\}$. By definition we know that $v_1, ..., v_d \in \partial C$. Suppose $e \in C$ and the nodes $v_1, ..., v_d$ can be partitioned into t non-empty set $S_1, ..., S_t$ according to their labels. Without loss of generality, let $v_1 \in S_1$. Then after CE of e we know that the edges $(v_1, v), v \in S_j, j \in \{2, 3, ..., t\}$ will be in the set $C^{(ce)}$. By definition of $C^{(ce)}$, we know that all $v \in S_j, j \in \{2, 3, ..., t\}$ will be in the cut set $\partial C^{(ce)}$. We can repeat the same argument for all nodes in S_1 and know that $S_1 \subset \partial C^{(ce)}$. In the end, we can show that $\forall v \in e$, $v \in \partial C^{(ce)}$. By definition we also have $\forall v \in e$, $v \in \partial C$. Therefore, we claim that $\partial C = \partial C^{(ce)}$ and furthermore that $|\partial C|= |\partial C^{(ce)}|$.

A.2.2 Proof for the inequality

Now, we prove that $\min(|C|, |\partial C|) \leq \min(|C^{(ce)}, |\partial C^{(ce)}|)$. As above, we have proved $|\partial C|= |\partial C^{(ce)}|$. The case when $|\partial C^{(ce)}|\leq |C^{(ce)}|$ is an easy case. So, we only need to prove for the case when $|\partial C^{(ce)}|> |C^{(ce)}|$. We claim that if $|\partial C^{(ce)}|> |C^{(ce)}|$, then $|C|\leq |C^{(ce)}|$, which is proved as follows.

Let us first introduce an auxiliary graph G' that can be useful in the proof. $G' = (\partial C^{(ce)}, C^{(ce)})$ is a subgraph of $G^{(ce)}$ with the node set $\partial C^{(ce)}$ and the edge set $C^{(ce)}$. In the following, we show that when $|\partial C^{(ce)}|> |C^{(ce)}|$, then it is impossible for G' to have any cliques of size greater than or equal to 3. Note that by the definition of $C^{(ce)}$ and $\partial C^{(ce)}$, the auxiliary graph G' is connected. Moreover, as for the condition $|\partial C^{(ce)}|> |C^{(ce)}|$, we know that the average degree of G' is strictly less then 2. This is because

$$2 > \frac{2|C^{(ce)}|}{|\partial C^{(ce)}|} = \frac{\sum_{v \in \partial C^{(ce)}} d_v}{|\partial C^{(ce)}|},$$

where d_v is the degree of node v in G'. Hence it is impossible to have any cliques of sizes that are greater than or equal to 3 in G'.

By using the above observation and the definition of clique expansion, we know that when $|\partial C^{(ce)}|> |C^{(ce)}|$, all hyperedges in C are actually edges. Equivalently, we have $C = C^{(ce)}$, which implies $|C|= |C^{(ce)}|< |\partial C^{(ce)}|$. This concludes the proof.

By the end of this subsection, we would like to show that it is possible to have $\min(|C|, |\partial C|) < \min(|C^{(ce)}|, |\partial C^{(ce)}|)$ for some hypergraphs. Let C contain only one hyperedge e such that $|e|= 4$. Then it is obvious to see that $1 = |C|< |C^{(ce)}|= 6$ and $|\partial C^{(ce)}|= |\partial C|= 4$. Hence in this special example we have $\min(|C|, |\partial C|) < \min(|C^{(ce)}|, |\partial C^{(ce)}|)$.

A.3 Proof of Theorem 2.3.2

Before we start our proof, we need to prepare preliminary results. The first one is Theorem 3 in [15] that characterizes the theoretical performance of Algorithm 2 in [15].

Theorem A.3.1 (Theorem 3 in [15])**.** Given a set of M points which can be partitioned into k clusters, the Algorithm 2 in [15] will return all clusters of size at least $\frac{64k\log M}{(1-2p)^4}$ with probability at least $1 - \frac{2}{M}$. The corresponding query complexity is $O(\frac{Mk^2\log M}{(1-2p)^4})$.

Basically we use this theorem to analyze Phase 1 of Algorithm 4. The next one is a lemma that characterizes a lower bound of the KL divergence of two Bernoulli distributions.

Lemma A.3.2. Let us denote $D(x||y)$ to be the KL divergence of two Bernoulli distributions with parameters $x, y \in [0, 1]$ respectively. We have

$$D(x||y) \geq \frac{(y-x)^2}{2\min\{x,\, y\}}. \tag{A.1}$$

Remark A.3.1. Note that the bound is tighter than directly using Pinsker's inequality [34] when $y \leq 1/8$.

Now we start to prove Theorem 2.3.2. First we will show that Phase 1 of Algorithm 4 will return the correct partition $S_1, ..., S_k$ with high probability. From Theorem A.3.1 we know that we have to ensure our sampled M points contain all underlying true clusters with size at least $O(\frac{Mk^2\log M}{(1-2p)^4})$. Since we sample these M points uniformly at random, $(S_1, ..., S_k)$ is the multivariate hypergeometric random vector with parameters $(n, np_1, ..., np_k, M)$ and $\forall i,\ p_i = \frac{|\{v\in V|f(v)=i\}|}{n}$. It is well known that when $M \leq n/2$, the tail bound

for the multivariate hypergeometric distribution is [35, 36],

$$\begin{aligned}
&\mathbb{P}(S_i \leq M(p_i - \frac{p_i}{2})) \leq \exp(-MD(\frac{p_i}{2}||p_i)) \\
&\leq \exp(\frac{-Mp_i}{8}) \\
&\Rightarrow \mathbb{P}(S_i \leq \frac{M\beta}{2}) \leq \exp(\frac{-M\beta}{8}),
\end{aligned} \tag{A.2}$$

where we use Lemma A.3.2 for the second inequality. For the case $M \geq n/2$, we could apply trick of symmetry [35–37] and have

$$\begin{aligned}
&\mathbb{P}(S_i \leq M(p_i - \frac{p_i}{2})) \\
&\leq \exp(-(n-M)D(p_i + \frac{p_i M}{2(n-M)}||p_i)) \\
&\leq \exp(-(n-M)\frac{(\frac{p_i M}{2(n-M)})^2}{p_i(2 + \frac{M}{n-M})}) \\
&= \exp(-\frac{p_i M^2}{4(2n-M)}) \\
&\leq \exp(-\frac{Mp_i}{12}),
\end{aligned}$$

where the second inequality is via Lemma A.3.2 and the last inequality uses the assumption $M \geq n/2$. Hence, for all $M \leq n$, we have

$$\mathbb{P}(S_i \leq \frac{M\beta}{2}) \leq \exp(\frac{-M\beta}{12}). \tag{A.3}$$

Since we need (A.3) to hold for all i, we apply the union bound over all k events which gives

$$\mathbb{P}(\bigcap_{i=1}^{k}\{S_i \geq \frac{M\beta}{2}\}) \geq 1 - k\exp(\frac{-M\beta}{12}). \tag{A.4}$$

Now, we need M to be large enough such that $\frac{M\beta}{2}$ meets the requirement of Theorem A.3.1. Moreover, we also need M to be large enough such that this event holds with probability at least $1 - \frac{\delta}{4}$. For the first requirement, we have

$$\frac{M\beta}{2} \geq \frac{64k\log M}{(2p-1)^4} \Rightarrow \frac{M}{\log M} \geq \frac{128k}{\beta(2p-1)^4}.$$

This is exactly our first requirement on M in (2.4). For the high probability

requirement, we have

$$k \exp(\frac{-M\beta}{12}) \leq \frac{\delta}{4} \Rightarrow M \geq \frac{12}{\beta} \log \frac{4k}{\delta}.$$

This is exactly the second requirement on M in (2.4). Moreover, we also need Algorithm 2 of [15] to successfully recover all the true clusters with probability at least $1 - \frac{\delta}{4}$, and thus we have

$$\frac{2}{M} \leq \frac{\delta}{4} \Rightarrow M \geq \frac{8}{\delta}.$$

This is exactly the third requirement on M in (2.4).

Now assume that Algorithm 2 of [15] indeed returns all true clusters. We will analyze Phase 2. Start from assuming all S_i's are correctly clustered. Then for any new node v, from the algorithm we designed we will query for comparing v with all the M nodes that have been clustered. Before we continue, let us introduce some error events which are useful for the following analysis. Let $Er^{(i)}$ be the event that a node with label i is incorrectly clustered by the normalized majority voting. Let $Er_j^{(i)} = \{\frac{M_j}{|S_j|} > \frac{M_i}{|S_i|}\}$, for $j \neq i$, where M_j is the number of nodes in S_j that respond positively to the pairwise comparisons with node v. Note that we have $M_j \sim Bin(p, |S_j|)$ for $j \neq i$ and $M_i \sim Bin(1-p, |S_i|)$. All these M_l's are mutually independent. We start from analyzing the normalized majority voting for the unlabeled node v. Then we have

$$\mathcal{O}_p(v, u) \sim Ber(1-p) \; \forall u \in S_i;$$
$$\mathcal{O}_p(v, u) \sim Ber(p) \; \forall u \notin S_i,$$

where we recall that $\mathcal{O}_p(x, y)$ is the query answer for the point pair (x, y) from the noisy oracle $\mathcal{O}_p$. So the error probability $\mathbb{P}(Er^{(i)})$ that we misclassify the point v can be upper bounded by

$$\mathbb{P}(Er^{(i)}) \leq (k-1) \max_{j \neq i} \mathbb{P}(Er_j),$$

where we used the union bound. Moreover, we can upper bound $\mathbb{P}(Er_j^{(i)})$ as

follows (recall that $p < 1/2$):

$$\begin{aligned}\mathbb{P}(Er_j^{(i)}) &= \mathbb{P}(\frac{M_j}{|S_j|} > \frac{M_i}{|S_i|}) \\ &\leq \mathbb{P}(\frac{M_j}{|S_j|} \geq \frac{1}{2}) + \mathbb{P}(\frac{1}{2} > \frac{M_j}{|S_j|}).\end{aligned}$$

Denote $\lambda = \frac{1}{2} - p > 0$. So we have $\frac{1}{2} = \lambda + p = \bar{p} - \lambda$ where $\bar{p} = 1 - p$. Hence by Chernoff's bound the first term can be upper bounded by

$$\mathbb{P}(\frac{M_j}{|S_j|} \geq \frac{1}{2}) \leq \exp(-|S_j| \cdot D(p + \lambda || p)),$$

and similarly the second term can be upper bounded by

$$\mathbb{P}(\frac{1}{2} > \frac{M_i}{|S_i|}) \leq \exp(-|S_i| \cdot D(\bar{p} - \lambda || \bar{p})).$$

Hence we have

$$\begin{aligned}\mathbb{P}(Er^{(i)}) \leq (k-1)[&\max_{j \neq i} \exp(-|S_j| \cdot D(p + \lambda || p)) \\ &+ \exp(-|S_i| \cdot D(\bar{p} - \lambda || \bar{p}))].\end{aligned}$$

Recall that from (A.4), we have $\min_{i \in [k]} |S_i| \geq \frac{M\beta}{2}$ with probability at least $1 - \frac{\delta}{4}$. Moreover, we observe that $D(0.5||p) = \min\{D(p + \lambda||p), D(\bar{p} - \lambda||\bar{p})\}$ by the symmetry of KL-divergence for Bernoulli distribution. Thus, the error probability for any new point can be upper bounded as

$$\mathbb{P}(Er) \leq \max_i \mathbb{P}(Er^{(i)}) \leq 2(k-1) \exp(\frac{-M\beta D(0.5||p)}{2}).$$

Note that from Theorem 2.2.5 we will need to query $\mathcal{Q}^*(\frac{\delta}{4})$ nodes in the aggressive search phase if we want the exact result to hold for probability at least $1 - \frac{\delta}{4}$ in noiseless case. Hence by using the union bound, the error probability for exact recovery of these $\mathcal{Q}^*(\frac{\delta}{4})$ points is upper bounded by

$$2\mathcal{Q}^*(\frac{\delta}{4})(k-1) \exp(\frac{-M\beta D(0.5||p)}{2}).$$

Requiring this to be smaller than $\frac{\delta}{4}$, then we have

$$M \geq \frac{2}{\beta D(0.5||p)} \log(\frac{8(k-1)\mathcal{Q}^*(\frac{\delta}{4})}{\delta}).$$

This is exactly the forth requirement on M in (2.4). Further, via the union bound, the overall algorithm will succeed with probability at least $1-\delta$. Note that if we have exact recovery on these $\mathcal{Q}^*(\frac{\delta}{4})$ nodes, then we can indeed find the cut set C by Theorem 2.2.5, which concludes the proof.

REFERENCES

[1] M. Bilgic, L. Mihalkova, and L. Getoor, "Active learning for networked data," in *Proceedings of the 27th International Conference on Machine Learning*, June 2010, pp. 79–86.

[2] J. He and J. G. Carbonell, "Nearest-neighbor-based active learning for rare category detection," in *Advances in Neural Information Processing Systems*, Dec. 2008, pp. 633–640.

[3] S. Towslee, "How blockchain and RFID will change supply chain management," *Medium*, Aug. 2018.

[4] S. Taoufik, P. Dherbecourt, A. El Oualkadi, and F. Temcamani, "Reliability and failure analysis of UHF RFID passive tags under thermal storage," *IEEE Trans. Device Mat. Rel.*, vol. 17, no. 3, pp. 531–538, Sep. 2017.

[5] A. Guillory and J. A. Bilmes, "Label selection on graphs," in *Advances in Neural Information Processing Systems*, Dec. 2009, pp. 691–699.

[6] Q. Gu and J. Han, "Towards active learning on graphs: An error bound minimization approach," in *IEEE 12th International Conference on Data Mining, 2012*, Dec. 2012, pp. 882–887.

[7] N. Cesa-Bianchi, C. Gentile, F. Vitale, and G. Zappella, "Active learning on trees and graphs," *arXiv preprint arXiv:1301.5112*, 2013.

[8] G. Dasarathy, R. Nowak, and X. Zhu, "S2: An efficient graph based active learning algorithm with application to nonparametric classification," in *Conference on Learning Theory*, 2015.

[9] S. Banker, "Blockchain gains traction in the food supply chain," *Forbes Mag.*, July 2018.

[10] S. Lewis, "Are you prepared for the high cost of a food recall?" *Food Online*, May 2015.

[11] D. Moore, C. Shannon, and K. Claffy, "Code-Red: a case study on the spread and victims of an internet worm," in *Proc. 2nd ACM SIGCOMM Workshop Internet Measurement*, Nov. 2002, pp. 273–284.

[12] R. Richardson and C. Director, "CSI computer crime and security survey," *Computer Security Institute*, vol. 1, pp. 1–30, 2008.

[13] D. Shah and T. Zaman, "Rumors in a network: Who's the culprit?" *IEEE Trans. Inf. Theory*, vol. 57, no. 8, pp. 5163–5181, July 2011.

[14] K. Zhu and L. Ying, "Information source detection in the SIR model: A sample-path-based approach," *IEEE/ACM Trans. Netw.*, vol. 24, no. 1, pp. 408–421, Feb. 2016.

[15] A. Mazumdar and B. Saha, "Clustering with noisy queries," in *Advances in Neural Information Processing Systems*, Dec. 2017, pp. 5788–5799.

[16] D. Prelec, H. S. Seung, and J. McCoy, "A solution to the single-question crowd wisdom problem," *Nature*, vol. 541, no. 7638, p. 532, 2017.

[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, 2009.

[18] J. Gao, Q. Zhao, W. Ren, A. Swami, R. Ramanathan, and A. Bar-Noy, "Dynamic shortest path algorithms for hypergraphs," *IEEE/ACM Transactions on Networking*, vol. 23, no. 6, pp. 1805–1817, 2015.

[19] D. J. Watts and S. H. Strogatz, "Collective dynamics of small-worldnetworks," *Nature*, vol. 393, no. 6684, p. 440, 1998.

[20] M. Hein, S. Setzer, L. Jost, and S. S. Rangapuram, "The total variation on hypergraphs-learning on hypergraphs revisited," in *Advances in Neural Information Processing Systems*, Dec. 2013, pp. 2427–2435.

[21] P. Li and O. Milenkovic, "Submodular hypergraphs: p-Laplacians, Cheeger inequalities and spectral clustering," in *Proceedings of the 35th International Conference on Machine Learning*, June 2018, pp. 3014–3023.

[22] P. Purkait, T.-J. Chin, A. Sadri, and D. Suter, "Clustering with hypergraphs: the case for large hyperedges," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 9, pp. 1697–1711, 2017.

[23] D. Zhou, J. Huang, and B. Schölkopf, "Learning with hypergraphs: Clustering, classification, and embedding," in *Advances in Neural Information Processing Systems*, Dec. 2007, pp. 1601–1608.

[24] S. Agarwal, K. Branson, and S. Belongie, "Higher order learning with graphs," in *Proceedings of the 23rd International Conference on Machine Learning*, June 2006, pp. 17–24.

[25] I. Chien, C.-Y. Lin, I. Wang et al., "On the minimax misclassification ratio of hypergraph community detection," *arXiv preprint arXiv:1802.00926*, 2018.

[26] P. Li and O. Milenkovic, "Inhomogeneous hypergraph clustering with applications," in *Advances in Neural Information Processing Systems*, Dec. 2017, pp. 2308–2318.

[27] R. Vidal, "Subspace clustering," *IEEE Signal Processing Magazine*, vol. 28, no. 2, pp. 52–68, 2011.

[28] R. Tron and R. Vidal, "A benchmark for the comparison of 3-d motion segmentation algorithms," in *IEEE Conference on Computer Vision and Pattern Recognition, 2007*, June 2007, pp. 1–8.

[29] J. P. Costeira and T. Kanade, "A multibody factorization method for independently moving objects," *International Journal of Computer Vision*, vol. 29, no. 3, pp. 159–179, 1998.

[30] A. L. Horn and H. Friedrich, "Locating the source of large-scale diffusion of foodborne contamination," *Journal of the Royal Society Interface*, vol. 16, no. 151, Feb. 2019.

[31] J. G. Kemeny and J. L. Snell, *Finite Markov Chains*. New York: Springer, 1983.

[32] F. Le Gall, "Powers of tensors and fast matrix multiplication," in *Proc. 39th Int. Symp. Symbolic Algebraic Comput. (ISSAC '14)*, July 2014, pp. 296–303.

[33] J. Leskovec and J. J. McAuley, "Learning to discover social circles in ego networks," in *Advances in Neural Information Processing Systems*, Dec. 2012, pp. 539–547.

[34] M. S. Pinsker, *Information and Information Stability of Random Variables and Processes*. Izv. Akad. Nauk, 1960.

[35] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 13–30, 1963.

[36] M. Skala, "Hypergeometric tail inequalities: Ending the insanity," *arXiv preprint arXiv:1311.5939*, 2013.

[37] R. J. Serfling, "Probability inequalities for the sum in sampling without replacement," *The Annals of Statistics*, pp. 39–48, 1974.