HS^2: Active learning over hypergraphs with pointwise and pairwise queries

I (Eli) Chien
Department ECE,UIUC

Huozhi Zhou
Department ECE,UIUC

Pan Li
Department ECE,UIUC

Abstract

We propose a hypergraph-based active learning scheme which we term HS^2; HS^2 generalizes the previously reported algorithm S^2 originally proposed for graph-based active learning with pointwise queries [1]. Our HS^2 method can accommodate hypergraph structures and allows one to ask both pointwise queries and pairwise queries. 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 generalized settings. Both the theoretical and empirical results show that HS^2 requires a significantly fewer number of queries than S^2 when one uses S^2 over a graph obtained from the corresponding hypergraph via clique expansion.

1 Introduction

Active learning is useful for many machine learning applications where the acquisition of labeled data is expensive and time consuming [2]. In this setting, the learner aims to query for as few labels of data points as possible while still guaranteeing a desired level of label prediction accuracy.

Graph-based active learning (GAL) refers to the particular case when graphs can be used to represent the data points. Such graphs may either come from real-life networks, e.g. social networks [3], or some transformation based on data points, e.g. nearest neighbor graphs [4]. Due to the prevalence of graph-structured data, GAL has attracted significant attention in the recent research literature. Most previous works on GAL shared a similar approach: the nodes selected for labeling are determined by minimizing some assumptive empirical error [5] or upper bound of an empirical error [6, 7, 8]. Recently, Dasarathy et al. studied the GAL problem from a substantially different angle [1]: They attempted to directly detect the boundaries of different classes over the graph, which further lead to the classification of nodes. Their approach, termed S^2, benefits from tracking labels of the midpoint of the shortest path among all paths whose ending nodes are with different labels. Surprisingly, S^2 is shown to be nearly min-max optimal for the non-parametric setting of active learning problems [1].

All above works for GAL depend on a *pointwise oracle*, that is, each response of a query leading to the label of one vertex. However, recent works have pointed out that humans are better at making comparison, and therefore, a *pairwise oracle*, whose response is of the form "nodes u and v (do not) belong to in the same class", appears more practical than the pointwise oracle [9, 10, 11, 12]. Mazumadar and Saha [13] proposed a GAL algorithm that uses pairwise oracles. However, their algorithm strongly depends on the assumption that the underlying graph is generated by a stochastic block model (SBM) [14].

Graphs essentially capture pairwise relations between data points. Recently, many machine learning and data mining applications have found that hypergraphs modeling high-order relations may lead to better learning performance than traditional graph-based models [15, 16, 17]. For example, in subspace clustering, a fit to d-dimensional subspace can only be evaluated over at least $d+1$ data points [18, 19]; in hierarchical species classification over a foodweb, a carbon-flow unit based on four species appears to be most predictive [19]. Although some unsupervised or semisupervised learning algorithms over hypergraphs have been reported in [20, 21, 22], few works targeted the setting of hypergraph-based active learning (HAL). Intuitively, to solve HAL problems, one may first "project" hypergraphs into graphs by replacing hyperedges with cliques (typically this is a procedure termed clique ex-

Proceedings of the 22[nd] International Conference on Artificial Intelligence and Statistics (AISTATS) 2019, Naha, Okinawa, Japan. PMLR: Volume 89.

pansion (CE) [18, 15]) and then use traditional GAL methods. However, a large number of works have demonstrated that CE causes distortion and leads to undesired learning performance [23, 24]. To the best of our knowledge, the approach proposed by Guillory et al. [25] is the only work that may directly handle HAL. This method follows the traditional rule of GAL [6] that attempts to minimize an upper bound one the prediction error and only works for a pointwise oracle.

Here, we focus on HAL problems. Following the rule used in S^2 [1], we develop several active learning algorithms, termed with the prefix "HS^2" , which are compatible with multiple classes and with pointwise/pairwise oracles in the hypergraph setting. Our contributions are as follows. First, for the setting with a pointwise oracle, we provide a more general algorithm with tighter analysis compared to [1]: We consider the k-class ($k \geq 2$) setting instead of the two-class one considered in the original S^2 algorithm. We define novel complexity parameters that can handle the complex interaction between k-ary labels and hyperedges. We derive a tighter bound of query complexity, which, as a by-product, justifies that the proposed algorithm indeed requires fewer queries than a simple combination of CE and S^2. Second, for the setting with a pairwise oracle, we develop the first model-free HAL/GAL algorithm: Our algorithm does not need any generative assuptions on the graphs/hypergraphs like SBM [13]; Moreover, the corresponding analysis for the pairwise-oracle setting is novel.

The paper is organized as follows: In Section 2, we introduce the notation and our problem formulation. In Section 3, we focus on the case of the pointwise oracle and theoretically demonstrate the superiority of HS^2 over a combination of CE and S^2. In Section 4 we focus on the case of the pairwise oracle. In Section 5 we present experiments for both synthetic and real-world data to verify our theoretical findings. Due to the page limitation, we defer the missing proofs to the supplement.

2 Problem formulations

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.

As 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 node (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. The original S^2 paper considers an simpler noise model [1], 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 [10, 26], such a method intrinsically introduces bias and thus majority voting may even increase the error. Therefore, we consider a more realistic model used in [10]. 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.

3 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 [1] to the hypergraph setting. HS^2-point is similar to S^2, in so far that 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 [1], 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 3.1 (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 $e_i \cap e_{i+1} \neq \emptyset \quad \forall i \in [l-1]$.

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

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 $MSSP(G, L)$. 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. 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.

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 BFS algorithm [27]. A more efficient way to search the shortest path in a dynamic hypergraph is described in [28]. Since we focus on query complexity, discussion of the time complexity of the algorithms is outside the scope of the paper.

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 $P_{u,v}$.(= ∞ 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

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

Definition 3.2 (Balancedness)**.** We say that G is β-balanced if $\beta = \min_{i \in [k]} \frac{|V_i|}{n}$.

Definition 3.3 (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 hyper-

edges as $\Delta : C \times C \to \mathbb{N} \cup \{0, \infty\}$: for $e_1, e_2 \in C$,

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

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 $\Delta(e, e') \le r$. 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 3.4 (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.

κ-clusteredness indicates the cut hyperedges in one cut component should not be κ 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 $O(\log n)$ at most [29].

The novel part of HS^2-point is that we propose Definition 3.3 and Definition 3.4, which properly generalizes the parametric system of S^2 [1] to hypergraphs and leads to the following theoretical estimation of query complexity.

Theorem 3.5. 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

$$\begin{aligned}\mathcal{Q}^*(\delta) &\triangleq \frac{\log(1/(\beta\delta))}{\log(1/(1-\beta))} + m(\lceil \log_2(n) - \log_2(\kappa) \rceil)\\ &+ \min(|\partial C|, |C|)(\lceil \log_2(\kappa) \rceil + 1) \end{aligned} \quad (2)$$

Note that Theorem 3.5 not only generalizes Theorem 3 from [1] 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 [23, 24, 30]. 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 [1]. We will illustrate this point in the next subsection.

3.1 Comparison with clique expansion

Clique expansion (CE) is a frequently used tool for learning tasks over hypergraphs [15, 16, 21, 19]. CE refers to the procedure that transforms hypergraphs into graphs by expanding hyperedges into cliques. Based on the graph obained 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 3.6. $G^{(ce)}$ 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 $\min(|C|, |\partial C|) \le \min(|C^{(ce)}|, |\partial C^{(ce)}|)$.

As graphs are special case of hypergraphs, Theorem 3.5 can be used to characterize the query complexity of S^2 over $G^{(ce)}$. For this purpose, recall the parameters in Theorem 3.5 that determine the query complexity. Combining them with Proposition 3.6, 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 $|C| \le |C^{(ce)}|$. To see the benefit of HS^2-point more clearly, consider the example in Figure 1. Once HS^2-point finds and removes

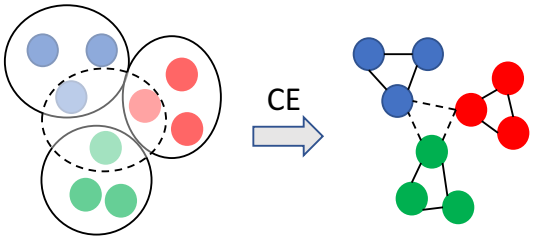

Figure 1: An example of clique expansion. Left: the orginal hypergraph G with 4 hyperedges; Right: the clique-expanded graph $G^{(ce)}$. The colors of nodes identify the labels and the dashed hyperedges/edges are cut hyperedges/edges.

the cut hyperedge of G, the correct partition of V is learnt. 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 3.1. 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 [1] does not reflect such a parametric dependence.

Remark 3.2. Note that in the example in Figure 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 3.6 still holds. This example shows that it is non-trivial to prove Proposition 3.6.

4 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.

4.1 Noiseless case

We start by introducing the setting with a noiseless pariwise 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 3.5, 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 4.1. 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.,

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

4.2 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 similar strategy as used in Algorithm 2 of [10] 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.

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 [10] 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

We know describe the query complexity of Algorithm 4.

Theorem 4.2. 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

$$\mathcal{Q}^*(\delta/4)M + \frac{128Mk^2\log M}{(2p-1)^4} \tag{3}$$

where $\mathcal{Q}^*(\delta)$ is defined in (2), and M is an integer satisfying

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

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 4.2. The complete proof is postponed to the supplement.

Proof. (sketch) In Phase 2, we expect to select $\mathcal{Q}^*(\delta_1)$ nodes for labeling, according to Theorem 3.5. 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

$$\min_{i\in[k]}|S_i| \geq \frac{1}{D(0.5||p)}\log\frac{2(k-1)\mathcal{Q}^*(\delta_1)}{\delta_2}. \tag{5}$$

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

$$\min_{i\in[k]}|S_i| \geq \frac{\beta M}{2}. \tag{6}$$

Combining (5) and (6) gives the fourth inequality in (4). Moreover, we also need to cluster these S_i correctly via Algorithm 2 in [10], which requires the first three constrains in (4) and the additional $\frac{128Mk^2\log M}{(2p-1)^4}$ queries according to Theorem 3 in [10]. This gives the formulas in Theorem 4.2. □

Remark 4.1. Suppose that the parameters (p, k, δ, β) are constants. Then, the fourth requirement of M in (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 3.5 and Corollary 4.1, 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 4.3. 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
$$\begin{aligned}\mathcal{Q}_1^* &= m(\lceil \log_2(n) - \log_2(\kappa) \rceil) \\ &\quad + \min(|\partial C|, |C|)(\lceil \log_2(\kappa) \rceil + 1),\end{aligned}$$

and now M is the smallest integer satisfying

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

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 4.1 and Theorem 4.2 again indicate, the query complexity depends on $\min\{|C|, |\partial C|\}$; By using Proposition 3.6, we can again demonstrate the superiority of our proposed approaches over CE-based methods.

5 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 [1] and EBM [7], 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 3.1. 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 [31]. We use the popular benchmark — the Hopkins 155 dataset [32] to evaluate the performance. As mentioned in [33, 30], 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.

5.1 Synthetic data

For the synthetic data, we investigate the effects of the scale of hypergraphs n, 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. 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). In addition, we also vary the size of hyperedges and the number of cut edges. The results are shown in Figure 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 doesn't have to. All these results suggest that our HS^2 is better than the CE+S^2 approach.

For the experiments on pairwise noiseless queries, the results are shown in Figure 4. Again, our HS^2-pair algorithm outperforms the naive combination of CE and S^2-pair. Different from 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

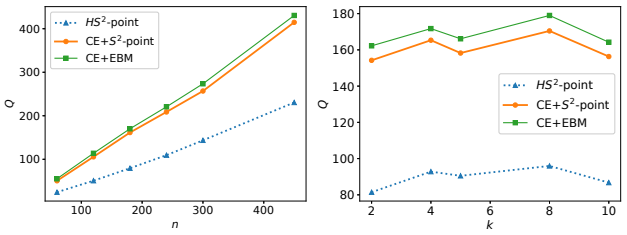

Figure 2: Simulation results with pointwise oracles over synthetic data. Left: query complexity vs scale of hypergraphs n with fixed $k = 3$; Right: query complexity vs the number of classes k with fixed $n = 200$.

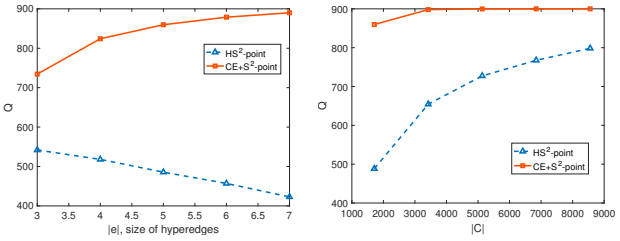

Figure 3: Simulation results with pointwise oracles over synthetic data. Left: query complexity vs size of hyperedges with fixed $k = 3$, $n = 900$; Right: query complexity vs the number cut edge $|C|$ with fixed $k = 3$, $n = 900$.

larger hypergraph, 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 1. As expected, HS^2 is better than CE+S^2 in terms of query complexity.

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

Table 1: Query complexity with noisy pairwise oracle on synthetic graphs.

5.2 Real world application

We test the algorithms on 4 checker board 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 [30] 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

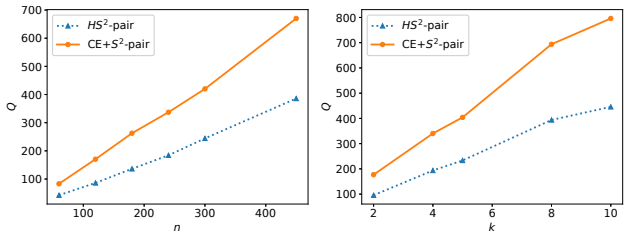

Figure 4: Simulation results with pairwise oracles over synthetic data. Left: query complexity vs scale of hypergraphs n with fixed $k = 3$; Right: query complexity vs the number of classes k with fixed $n = 200$.

total inner-cluster sample. 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 5. We can see that indeed HS^2 needs much less queries than CE+S^2, which matches our theoretical and synthetic results.

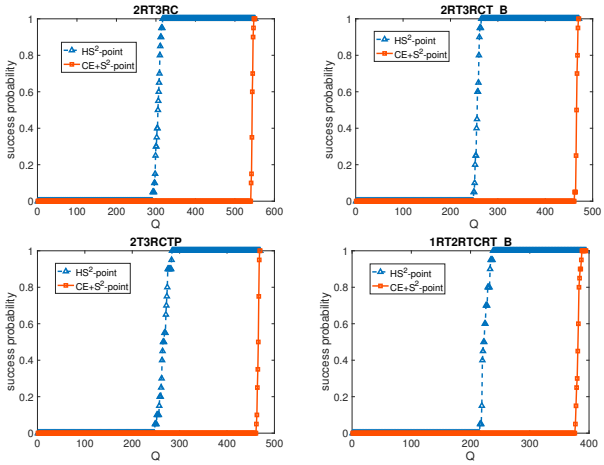

Figure 5: Results for the experiments on the Hopkins 155 dataset. The title for each subfigure describes how the correponding data is obtained. For example, 2RT means that object 2 translates and rotates on the same axis. Each task contains about 400 to 500 points.

Acknowledgement

This work was supported in part by the grants NIH 3U01CA198943-02S1, NSF 0939370 Emerging Frontiers of Science of Information and NSF CCF 15-26875. This work was also 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. We greatly thank to Dr. Olgica Milenkovic and Dr. Lav R. Varshney for their help on writing.

References

[1] 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.

[2] D. Cohn, L. Atlas, and R. Ladner, "Improving generalization with active learning," *Machine learning*, vol. 15, no. 2, pp. 201–221, 1994.

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

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

[5] X. Zhu, J. Lafferty, and Z. Ghahramani, "Combining active learning and semi-supervised learning using gaussian fields and harmonic functions," in *ICML 2003 workshop on the continuum from labeled to unlabeled data in machine learning and data mining*, vol. 3, 2003.

[6] A. Guillory and J. A. Bilmes, "Label selection on graphs," in *Advances in Neural Information Processing Systems*, 2009.

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

[8] N. Cesa-Bianchi, C. Gentile, F. Vitale, and G. Zappella, "Active learning on trees and graphs," in *Annual Conference on Learning Theory*, 2010.

[9] H. Ashtiani, S. Kushagra, and S. Ben-David, "Clustering with same-cluster queries," in *Advances in Neural Information Processing Systems*, 2016.

[10] A. Mazumdar and B. Saha, "Clustering with noisy queries," in *Advances in Neural Information Processing Systems*, 2017.

[11] I. Chien, C. Pan, and O. Milenkovic, "Query k-means clustering and the double dixie cup problem," in *Advances in Neural Information Processing Systems*, 2018.

[12] C. E. Tsourakakis, M. Mitzenmacher, K. G. Larsen, J. Błasiok, B. Lawson, P. Nakkiran, and V. Nakos, "Predicting positive and negative links with noisy queries: theory & practice," *arXiv preprint arXiv:1709.07308*, 2017.

[13] A. Mazumdar and B. Saha, "Query complexity of clustering with side information," in *Advances in Neural Information Processing Systems*, 2017.

[14] P. W. Holland, K. B. Laskey, and S. Leinhardt, "Stochastic blockmodels: First steps," *Social networks*, vol. 5, no. 2, pp. 109–137, 1983.

[15] D. Zhou, J. Huang, and B. Schölkopf, "Learning with hypergraphs: Clustering, classification, and embedding," in *Advances in Neural Information Processing Systems*, 2007.

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

[17] A. R. Benson, D. F. Gleich, and J. Leskovec, "Higher-order organization of complex networks," *Science*, vol. 353, no. 6295, pp. 163–166, 2016.

[18] S. Agarwal, J. Lim, L. Zelnik-Manor, P. Perona, D. Kriegman, and S. Belongie, "Beyond pairwise clustering," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005.*, vol. 2. IEEE, 2005.

[19] P. Li and O. Milenkovic, "Inhomogeneous hypergraph clustering with applications," in *Advances in Neural Information Processing Systems*, 2017.

[20] I. Chien, C.-Y. Lin, and I.-H. Wang, "Community detection in hypergraphs: Optimal statistical limit and efficient algorithms," in *International Conference on Artificial Intelligence and Statistics*, 2018.

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

[22] P. Li, N. He, and O. Milenkovic, "Quadratic decomposable submodular function minimization," in *Advances in Neural Information Processing Systems*, 2018.

[23] 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*, 2013.

[24] P. Li and O. Milenkovic, "Submodular hypergraphs: p-laplacians, cheeger inequalities and spectral clustering," in *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[25] A. Guillory and J. Bilmes, "Active semi-supervised learning using submodular functions," in *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, 2011.

[26] 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.

[27] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[28] 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.

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

[30] 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.

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

[32] 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.* IEEE, 2007.

[33] 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.

[34] Wikipedia contributors, "Chernoff bound — Wikipedia, the free encyclopedia," 2018. [Online]. Available: https://goo.gl/CFJsvT

[35] M. S. Pinsker, *Information and information stability of random variables and processes*. Izv. Akad. Nauk, 1960.

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

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

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