HyperSearch: Prediction of New Hyperedges through Unconstrained yet Efficient Search

Hyunjin Choo[1], Fanchen Bu[1], Hyunjin Hwang[2] Young-Gyu Yoon[1], Kijung Shin[1,2]
[1]School of Electrical Engineering and [2]Kim Jaechul Graduate School of AI, KAIST, Republic of Korea
{choo, boqvezen97, hyunjinhwang, ygyoon, kijungs}@kaist.ac.kr

***Abstract*—Higher-order interactions (HOIs) in complex systems, such as scientific collaborations, multi-protein complexes, and multi-user communications, are commonly modeled as hypergraphs, where each hyperedge (i.e., a subset of nodes) represents an HOI among the nodes. Given a hypergraph, *hyperedge prediction* aims to identify hyperedges that are either missing or likely to form in the future, and it has broad applications, including recommending interest-based social groups, predicting collaborations, and uncovering functional complexes in biological systems. However, the vast search space of hyperedge candidates (i.e., all possible subsets of nodes) poses a significant computational challenge, making naïve exhaustive search infeasible. As a result, existing approaches rely on either heuristic sampling to obtain constrained candidate sets or ungrounded assumptions on hypergraph structure to select promising hyperedges.**

In this work, we propose HyperSearch, a search-based algorithm for hyperedge prediction that *efficiently* evaluates *unconstrained* candidate sets, by incorporating two key components: (1) an *empirically grounded* scoring function derived from observations in real-world hypergraphs and (2) an *efficient* search mechanism, where we derive and use an anti-monotonic upper bound of the original scoring function (which is not anti-monotonic) to prune the search space. This pruning comes with *theoretical guarantees*, ensuring that discarded candidates are never better than the kept ones w.r.t. the original scoring function. In extensive experiments on 10 real-world hypergraphs across five domains, HyperSearch consistently outperforms state-of-the-art baselines, achieving higher accuracy in predicting new (i.e., not in the training set) hyperedges.

***Index Terms*—Hypergraph, Hyperedge Prediction, Search**

I. Introduction

In many complex real-world systems, groups of entities engage simultaneously, forming *higher-order interactions* (HOIs), e.g., scientific collaborations (coauthorship among multiple researchers) [1], multi-protein complexes in biological systems [2], and multi-user communications [3].

Hypergraphs, which extend ordinary (pairwise) graphs by allowing hyperedges to connect multiple nodes, offer a natural framework for modeling HOIs. Unlike graphs, where an edge links exactly two nodes, hyperedges capture arbitrary-sized relationships, providing a more expressive representation of complex multi-way interactions. This enables the discovery of structural patterns and behaviors that are often obscured in pairwise frameworks [1], [4], [5].

Given a hypergraph, *hyperedge prediction* aims to identify hyperedges that are missing or to emerge in the future based on observed data. This task is essential for improving predictive capabilities in systems where HOIs play a crucial role. Hyperedge prediction has broader applications, including:

- **Group recommendation**: In social networks, hyperedge prediction enables the recommendation of groups, each consisting of users sharing common interests or behaviors, enhancing user experience in social media platforms and improving the effectiveness of targeted marketing [6], [7].
- **Collaboration prediction**: In academic and industrial networks, hyperedge prediction helps identify potential collaborations among researchers or companies with shared interests or expertise, optimizing team formation [8], [9].
- **Drug discovery**: In protein and gene networks, hyperedge prediction aids in forecasting functional protein complexes or interacting gene groups, providing insights into disease mechanisms and facilitating drug discovery [10], [11].

However, the problem of hyperedge prediction is computationally challenging. The number of potential hyperedges is $\mathcal{O}(2^n)$ for n nodes, making exhaustive enumeration infeasible for even medium-sized hypergraphs with hundreds of nodes.

While various methods have been proposed to efficiently identify promising hyperedges [12]–[16], they suffer from two fundamental limitations:

- **Constrained candidate sets**: Most deep learning-based methods (e.g., Hyper-SAGNN [13] and NHP [16]) frame hyperedge prediction as a binary classification task, distinguishing between existing (positive) and nonexistent (negative) hyperedges. Since enumerating all potential hyperedges is computationally infeasible, negative hyperedges are typically *sampled heuristically*. Consequently, the performance of such methods strongly depends on the quality of these samples, and how to sample desirable negative hyperedges is a challenging problem itself (see, e.g., AHP [15]). Alternatively, MHP [14] assumes a given query node set Q and predicts hyperedges by initializing hyperedges with nodes in Q and filling the missing nodes. However, the predictions by MHP are *constrained by the choice* of the query node set Q, and selecting Q is non-trivial in practice, limiting its flexibility and generalizability.
- **Unjustified structural assumptions**: HPRA [12] constructs candidate hyperedges by adding "similar" nodes based on resource allocation scores. However, these scores *strongly depends on observed structures* and assume specific *local connectivity patterns* (e.g., the similarity between two nodes is proportional to the number of neighbors they share), *without sufficient justification*. Such assumptions make the method less flexible and generalizable, particularly when discovering new hyperedges connecting nodes with low

structural similarity. Moreover, HPRA is limited to predicting hyperedges consisting only of nodes within the same connected component, as the similarity between nodes in different components is zero. This constraint limits its applicability in sparse or disconnected hypergraphs.

To address these limitations, we propose HyperSearch, a search-based algorithm for hyperedge prediction. HyperSearch is (1) *empirically justified*, i.e., grounded in real-world observations rather than ad-hoc heuristics, enabling generalization beyond observed structures, and (2) *unconstrained yet efficient*, i.e., efficiently explores the vast hyperedge search space without being constrained by predefined or sampled subsets.

- **Empirically justified scores based on observations**: The scoring function in HyperSearch is designed based on our observations in real-world datasets. Specifically, we observe that ground-truth hyperedges have *significant overlap* with observed hyperedges, and we thus assign each candidate hyperedge a score based on the number of observed hyperedges it significantly overlaps with. Additionally, when node features or timestamps are available, we incorporate node feature similarity to prioritize candidates consisting of nodes with similar features, and apply time weighting to emphasize recent hyperedges when computing overlaps, based on our observations.
- **Efficient search with an anti-monotonic upper bound**: The original scoring function is not anti-monotonic, i.e., a subset may have a lower score than its supersets, rendering naïve pruning strategies ineffective. To address this challenge, we derive an anti-monotonic upper bound of the original scoring function. This allows us to efficiently explore the search space with theoretical guarantees, ensuring that discarded candidates are never better than the kept ones w.r.t. the original scoring function.

We conduct extensive experiments on 10 real-world hypergraphs across five domains, which demonstrate the empirical superiority of HyperSearch:

- **Predictive accuracy**: HyperSearch consistently outperforms all baseline methods, including deep learning-based and rule-based ones, across diverse datasets, achieving higher accuracy in predicting new hyperedges.
- **Computational efficiency**: HyperSearch achieves faster runtime than deep learning-based methods in most cases, while maintaining near-linear scalability with respect to the input hypergraph size.
- **Component effectiveness**: Ablation studies empirically validate that each component of HyperSearch contributes meaningfully to performance.

Reproducibility: The code, datasets, and appendix are available at https://github.com/jin-choo/HyperSearch.

The remainder of this paper is organized as follows. In Sect. II, we review related work. In Sect. III, we introduce preliminary concepts and describe the datasets. In Sect. IV, we present empirical observations from real-world datasets. In Sect. V, we detail our proposed method. In Sect. VI, we review our experiments. In Sect. VII, we conclude the paper.

II. Related Work

A. Similarity-Based Methods

Early approaches to hyperedge prediction are mainly based on node similarity measures [17], [18]. They either extend node similarity measures to hypergraphs or project hypergraphs into pairwise graphs to apply node similarity measures. HPRA [12], for example, constructs hyperedges by selecting a seed node via preferential attachment (i.e., high-degree nodes are more likely to be chosen) and incrementally adding structurally similar nodes, guided by resource allocation scores.

Limitations. While such methods are computationally efficient, they primarily depend on local structural similarity (e.g., pairwise similarity) and are limited in capturing the complex higher-order dependencies inherent in real-world hypergraphs.

B. Deep Learning-Based Methods

To overcome the limitations of similarity-based approaches, deep learning models have been employed to learn higher-order representations in hypergraphs. DHNE [19] proposes an MLP-based framework for k-uniform hypergraphs. HyperSAGNN [13] extends this approach by incorporating a self-attention mechanism with graph neural networks, enabling non-k-uniform hyperedge prediction. NHP [16] further generalizes hyperedge prediction to directed hypergraphs and extends it to inductive settings.

Limitations. While these methods demonstrate strong performance, they typically rely on predefined candidate sets and require negative sampling strategies for effective training, as described below.

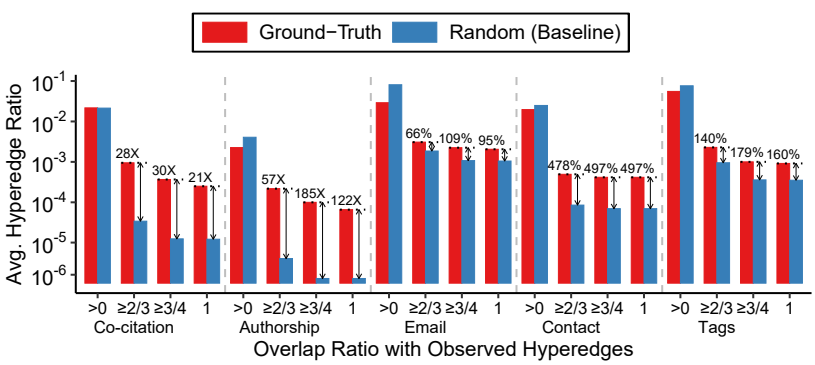
C. Negative Sampling on Hyperedges

Hyperedge prediction is commonly framed as a binary classification task, where the model distinguishes between observed (positive) and nonexistent (negative) hyperedges. Given the combinatorial explosion of potential hyperedges,[1] negative sampling plays a crucial role in defining meaningful training samples, and different negative sampling strategies introduce varying levels of difficulty, directly impacting prediction performance [20].

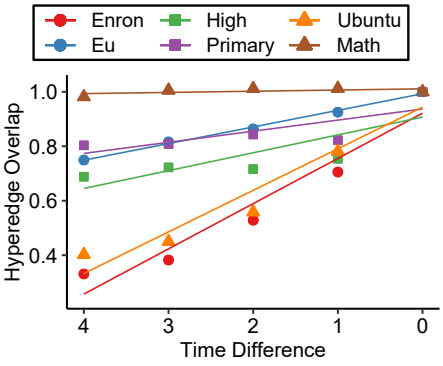
To address this challenge, AHP [15] introduces a generative adversarial training approach that dynamically generates negative samples during training, mitigating the dependence on heuristics. In contrast, MHP [14] proposes a novel masking-based approach that eliminates the need for explicit negative sampling by formulating hyperedge prediction as a masked node prediction task given a query node set and target size.

Limitations. However, they still depend on external constraints, such as the choice of query set in MHP, that inherently limit the candidate space. Consequently, their ability to explore hyperedge candidates remains limited, especially when compared to approaches like ours that search the full space without predefined candidates.

[1]The number of potential hyperedges is $\mathcal{O}(2^n)$ for n nodes.



(a) **Obs. 1. There is significant structural overlap between new hyperedges and observed hyperedges.**



(b) **Obs. 2. Overlap increases as the time difference decreases.**

Fig. 1. **Observations.** (a) The average proportion of ground-truth new hyperedges with overlap ratios with low threshold (> 0) is comparable to that of the random ones. However, for high thresholds ($\geq 2/3$, $\geq 3/4$, and 1), the gaps between ground-truth and random ones become significant, suggesting ground-truth new hyperedges are more likely to substantially overlap with existing hyperedges than random hyperedges are. (b) Structural overlap between hyperedges declines as the time gap increases, suggesting new hyperedges tend to overlap more substantially with recent existing ones than with earlier ones.

III. Preliminaries and Datasets

A. Concepts and Problem Statement

1) Hypergraphs: A *hypergraph* $H = (V, E)$ is defined by its *node set* $V = \{v_1, v_2, \ldots, v_{|V|}\}$ and *hyperedge set* $E = \{e_1, e_2, \ldots, e_{|E|}\}$. Each hyperedge e_j is a non-empty subset of nodes (i.e., $\emptyset \neq e_j \subseteq V$), and its *edge size* $|e_j|$ is the number of nodes in it. Each hyperedge e_j can also be associated with an *edge timestamp* t_j, if such information is available.

2) Problem Definition (Hyperedge Prediction):

- **Given:**
  1) a (partially) observed hypergraph $H = (V, E)$ (optionally with edge timestamps t_j's),
  2) the target number k of hyperedges to predict,
- **To find:** a set E' of k predicted new hyperedges (i.e., $E' \subseteq 2^V \setminus (E \cup \{\emptyset\})$ with $|E'| = k$),
- **Aim to:** Accurately predict hyperedges that are missing or are to emerge in the future in H.

B. Datasets

We use 10 real-world hypergraphs from five different domains, summarized in Table I.[2] Among them, the Email, Contact, and Tags datasets contain edge timestamps.

- **Co-citation (Citeseer, Cora)**: Each hyperedge consists of publications (nodes) cited by the same paper.
- **Authorship (Cora-A, DBLP-A)**: Each hyperedge consists of publications (nodes) by the same author.
- **Email (Eu, Enron)**: Each hyperedge consists of the sender (node) and all receivers (nodes) of an email.
- **Contact (High, Primary)**: Each hyperedge includes people (nodes) in a group interaction recorded by wearable sensors.
- **Tags (Math.sx, Ubuntu)**: Each hyperedge consists of the set of tags (nodes) added to the same question.

IV. Observations

This section highlights our key observations in real-world hypergraphs. We shall later design the scoring function in the proposed method (see Sect. V) based on these observations.

[2]Source 1: https://www.cs.cornell.edu/~arb/data
Source 2: https://github.com/HyunjinHwn/SIGIR22-AHP

TABLE I
Summary of Real-World Hypergraph Datasets.

Domain	Dataset	# Nodes	# Hyperedges	Timestamps
Co-citation	Citeseer	1,457	1,078	
	Cora	1,434	1,579	
Authorship	Cora-A	2,388	1,072	
	DBLP-A	39,283	16,483	
Email	Enron	143	10,883	✓
	Eu	998	234,760	
Contact	High	327	172,035	✓
	Primary	242	106,879	
Tags	Math.sx	1,629	822,059	✓
	Ubuntu	3,029	271,233	

A. Significant Overlap between Hyperedges

Our first observation is about the *overlap* between new hyperedges (which we aim to predict) and observed (training) hyperedges. We define the *overlap ratio* between a new hyperedge e' and an observed hyperedge e as $OverlapRatio(e', e) := |e' \cap e|/|e|$.

Setup: We randomly split the hyperedge set E in each dataset into two subsets: observed hyperedges E_1 (80% of E) and (ground-truth) new hyperedges E_2 (remaining 20% of E), using five random seeds. For each new hyperedge in E_2, we compute the proportion of observed hyperedges in E_1 with different overlap-ratio thresholds: > 0, $\geq 2/3$, $\geq 3/4$, and 1, and report the values averaged over all hyperedges in E_2.

As a baseline for comparison, we computed the same statistics on random hyperedges generated by a generalized Chung-Lu (CL) model [21], which preserves both node-degree and hyperedge-size distributions of the ground-truth E_2. Specifically, in each hyperedge, nodes are replaced with randomly drawn nodes, where each node is selected independently with probability proportional to its degree.

Observations: As shown in Fig. 1(a), the proportions of overlapping observed hyperedges for low threshold (> 0) show no significant difference between ground-truth hyperedges and the random ones. However, for higher thresholds ($\geq 2/3$, $\geq 3/4$, and $= 1$), ground-truth hyperedges exhibit significantly more overlap with observed hyperedges than the random

ones. This indicates that new hyperedges are more likely to substantially overlap with existing hyperedges than random hyperedges are, and this difference is statistically significant. [3]

B. Temporal Bias in Structural Overlap

Our second observation is about the impact of edge timestamps on structural overlap between hyperedges.

Setup: We partition the hyperedges E evenly into five equal-sized groups based on their timestamps, arranged in temporal order, where the first group G_1 contains the earliest hyperedges, the last group G_5 contains the most recent (latest) ones, etc. For each pair of groups G_i and G_j with $i < j$, we see the hyperedges in G_i as observed ones and the ones in G_j as new ones, and compute the average overlap ratio (see Sect. IV-A) between $e' \in G_j$ and $e \in G_i$. We further average the ratios w.r.t. the time gap $j - i$ (larger $j - i$ means longer time gaps).

Observations: As shown in Fig. 1(b), the structural overlap between hyperedges increases as the time difference decreases, indicating that new hyperedges tend to overlap more substantially with recent existing ones than with earlier ones.

V. Proposed Method: HyperSearch

In this section, we introduce the proposed method, HyperSearch, a search-based method for hyperedge prediction.

A. Overview (Alg. 1)

An overview of HyperSearch is presented in Algorithm 1. Given an observed hypergraph $H = (V, E)$ (optionally with edge timestamps t_j's), HyperSearch predicts new hyperedges by identifying the k high-scoring hyperedge candidates, determined by a scoring function designed based on our observations (see Sect. IV). Here, k is given as an input in the problem statement (see Sect. III-A2) and is not a hyperparameter of HyperSearch we need to fine-tune.

HyperSearch first determines the target number k_i of hyperedges to predict for each size i as $k_i := \text{round}\left(k\frac{|\{e\in E:|e|=i\}|}{|E|}\right)$, to match the original hyperedge size distribution in E. Then, HyperSearch performs a depth-first search (DFS) over the space of hyperedge candidates to identify the top-k_i hyperedges for each size, based on the scoring function. However, the original scoring function is not anti-monotonic (i.e., a subset may have a lower score than its supersets), making naïve pruning strategies ineffective. To address this challenge, HyperSearch employs an anti-monotonic upper bound of the original scoring function, which we derive. This allows HyperSearch to efficiently explore the search space with theoretical guarantees.

We shall describe two key components of HyperSearch: (1) scoring based on empirical patterns that accurately identifies promising hyperedges and (2) search with an anti-monotonic upper bound that enhances computational efficiency.

[3] At a higher threshold (spec., $\geq 2/3$), the Kolmogorov–Smirnov (KS) distance between the overlap-ratio distributions of ground-truth and random hyperedges is statistically significant ($KS = 0.26$, $p < 0.01$), confirming that the two distributions are clearly distinguishable with high confidence.

B. Scoring Based on Observations

We design the score for each candidate hyperedge e' based on our empirical observations in Sect. IV.

1) Prioritizing Candidates with High Overlap:

> **Key Idea From Obs. 1 (Sect. IV-A).** Ground-truth new hyperedges are likely to significantly overlap with many observed hyperedges. Therefore, in our scoring, we should prioritize candidates with high overlap.

Relaxed Overlap Count. Inspired by frequent itemset mining [22], we consider *relaxed overlap count*, controlled by three relaxation ratios between 0 and 1 (larger ratio means more relaxed): (1) node relaxation ratio ϵ_v, (2) hyperedge relaxation ratio ϵ_e, and (3) total relaxation ratio ϵ_t. Let E be the set of observed hyperedges, for each candidate hyperedge e', we say a subset $\widetilde{E} \subseteq E$ satisfies the criteria for e' w.r.t. the three relaxation ratios, if: (1) for each node v' in e', it is missing in at most ϵ_v proportion of the hyperedges in $\widetilde{E}$, i.e., $\forall v' \in e', |\{e \in \widetilde{E} : v' \notin e\}| \leq \epsilon_v|\widetilde{E}|$; (2) in each hyperedge $e \in \widetilde{E}$, at most ϵ_e proportion of the nodes in e' are missing, i.e., $\forall e \in \widetilde{E}, |v' \in e' : v' \notin e| \leq \epsilon_e|e'|$; and (3) the total missing occurrences of nodes in e' in $\widetilde{E}$ has proportion at most ϵ_t, i.e., $|v' \in e', e \in \widetilde{E} : v' \notin e| \leq \epsilon_t|e'||\widetilde{E}|$.

We let $\widetilde{E}(e', \epsilon_v, \epsilon_e, \epsilon_t)$ be the maximum subset satisfying the above conditions, and define the *relaxed overlap count* as $ovr(e', \epsilon_v, \epsilon_e, \epsilon_t) = |\widetilde{E}(e', \epsilon_v, \epsilon_e, \epsilon_t)|$. [4] The set $\widetilde{E}$ consists of hyperedges that collectively overlap with the nodes in e'. For example, if most hyperedges overlap only with a subset of e', they do not sufficiently support the likelihood of the entire e' and are therefore excluded from $\widetilde{E}$ due to the first constraint. Similarly, hyperedges that overlap only marginally with a small portion of e' are excluded due to the second constraint. If we use E instead of $\widetilde{E}$ for scoring (e.g., for the overlap count), however, such hyperedges, which do not meaningfully support the likelihood of e', increase its score, degrading the validity of the score.

The set $\widetilde{E}(e', \epsilon_v, \epsilon_e, \epsilon_t)$ can be computed by solving an Integer Linear Programming (ILP) problem. [5] Let the binary decision variable $x_j \in \{0, 1\}$ for each $e_j \in E$ indicate whether e_j is included in $\widetilde{E}$ ($x_j = 1$) or not ($x_j = 0$), and let $A \in \{0, 1\}^{|e'|\times|E|}$ be a binary matrix where $A_{i,j}$ indicates whether $v_i \notin e_j$ ($A_{i,j} = 1$) or not ($A_{i,j} = 0$). Then, the above constraints are formulated as follows:

1) **Node relaxation constraint:**

$$\sum\nolimits_{j=1}^{|E|} A_{i,j} \cdot x_j \leq \epsilon_v \cdot \sum\nolimits_{j=1}^{|E|} x_j \quad \forall i \in [|e'|]. \tag{1}$$

2) **Hyperedge relaxation constraint:**

$$\sum\nolimits_{i=1}^{|e'|} A_{i,j} \leq \epsilon_e \cdot |e'| \quad \forall j \text{ such that } x_j = 1. \tag{2}$$

[4] Considering such relaxation enhances the flexibility in scoring. Specifically, when relaxation is disabled, i.e., when $\epsilon_v = \epsilon_e = \epsilon_t = 0$, $ovr(e', 0, 0, 0)$ would be simply the number of observed hyperedges that are supersets of e'.

[5] In our implementation, we solve this ILP formulation using the SCIP solver [23] provided via Google OR-Tools.

3) **Total relaxation constraint:**

$$\sum_{i=1}^{|e'|}\sum_{j=1}^{|E|} A_{i,j}\cdot x_j \le \epsilon_t\cdot|e'|\cdot\sum_{j=1}^{|E|} x_j. \quad (3)$$

The objective then is to maximize the size of $\widetilde{E}$, i.e.,

$$\max\sum_{j=1}^{|E|} x_j. \quad (4)$$

Incorporating Overlap Ratio. Relaxed overlap count only accounts for the *number* of observed hyperedges that satisfy the criteria, so we further incorporate *overlap ratio* (see Sect. IV-A) into the scoring function to capture the degree of overlap of each observed hyperedge. The final score is

$$f_1(e') = \sum_{e\in\widetilde{E}(e',\epsilon_v,\epsilon_e,\epsilon_t)} \frac{|e'\cap e|}{|e|}. \quad (5)$$

2) Weighting More Recent Observed Hyperedges:

Key Idea From Obs. 2 (Sect. IV-B). Ground-truth new hyperedges are likely to overlap with more recent observed hyperedges than earlier ones. Therefore, in the scoring, we should add higher weights to more recent observed hyperedges.

Time weight. We introduce *time weight* that assigns greater significance to more recent hyperedges, when edge timestamps are available. Formally, the time weight for an observed hyperedge e is defined as: $\exp(\tau t_e)$, where τ is an adjustable parameter that determines the emphasis on recent hyperedges, and $t_e\in[0,1]$ is the normalized timestamp of e. With time weight included, the final score is

$$f_2(e') = \sum_{e\in\widetilde{E}(e',\epsilon_v,\epsilon_e,\epsilon_t)} \frac{|e'\cap e|}{|e|}\exp(\tau t_e). \quad (6)$$

3) Final Scoring Function: For datasets with edge timestamps, the final score for a hyperedge candidate e' is:

$$f_s(e') = f_2(e') = \sum_{e\in\widetilde{E}(e',\epsilon_v,\epsilon_e,\epsilon_t)} \frac{|e'\cap e|}{|e|}\exp(\tau t_e). \quad (7)$$

When edge timestamps are not available, $\exp(\tau t_e)$ is omitted, i.e., we use $f_s(e') = f_1(e') = \sum_{e\in\widetilde{E}(e',\epsilon_v,\epsilon_e,\epsilon_t)} \frac{|e'\cap e|}{|e|}$.

C. Pruning with Anti-Monotonic Upper Bound

As discussed in Sect. I, with an unconstrained candidate set, the naïve exhaustive enumeration of all $\mathcal{O}(2^{|V|})$ candidates and picking the best candidates is computationally infeasible in most cases. We aim to effectively and efficiently prune unconstrained candidate sets with theoretical guarantees.

For pruning set functions, a useful property is *anti-monotonicity*. A set function $f(\cdot)$ is anti-monotonic if $f(e_1)\ge f(e_2)$ for any $e_1\subseteq e_2$. If a function is anti-monotonic, when we identify a sub-optimal candidate e', we can prune all its supersets, because all its supersets are never better than itself. However, our original scoring function is not anti-monotonic in general (i.e., for arbitrary relaxation ratios). To this end, we aim to derive an *anti-monotonic upper bound* f_n of the original scoring function f_s. The key idea is, once we have f_n, we can use it as a *safe pruning criterion*: if a candidate

Algorithm 1: Overview of HyperSearch

Input: (1) $H=(V,E)$: observed hypergraph // Sect. III-A2
(optionally with edge timestamps t_j's)
(2) k: target number of hyperedges to predict // Sect. III-A2
(3) $(\epsilon_v,\epsilon_e,\epsilon_t)$: relaxation ratios // Sect. V-B1
(4) τ: growth constant for time weight // Sect. V-B2
Output: E': predicted set of new hyperedges

```
i_max ← max{|e| : e ∈ E}                                   // maximum hyperedge size
k_i ← round(k · |{e∈E:|e|=i}| / |E|), ∀i ≤ i_max            // targets; Sect. V-A
θ_i ← 0, ∀i ≤ i_max                                        // initialize top-k thresholds
E'_i ← ∅, ∀i ≤ i_max                                       // initialize outputs
foreach v ∈ V do
    S ← [(v, {v})]                                         // initialize DFS; Sect. V-D
    while S ≠ ∅ do
        pop (u, e') from S                                 // e': current candidate
        i ← |e'|                                           // i: current size
        if f_t(e') < θ_i then continue                     // prune e' and all its supersets if its score's upper bound f_t(e') does not exceed the top-k threshold; Sect. V-C
        calculate score f_s(e')                            // Sect. V-B
        if |E'_i| < k_i or f_s(e') ≥ θ_i then
            insert e' into E'_i and retain the top-k_i by score
            θ_i ← min_{e∈E'_i} f_s(e)                      // update θ_i; Sect. V-D
        if |e'| = i_max then continue                      // too large size
        foreach w ∈ V \ e' do
            push (w, e' ∪ {w}) into S                      // proceed DFS
E' ← ⋃_{i≤i_max} E'_i                                      // collect outputs
return E'
```

e' satisfies $f_n(e')<\theta$ for some pruning threshold θ, then all supersets e'' of e' can be pruned without loss of optimality:

$$f_s(e'') \overset{\text{upper bound}}{\le} f_n(e'') \overset{\text{anti-mono.}}{\le} f_n(e') < \theta, \forall e''\supseteq e'. \quad (8)$$

Specifically, we leverage the following anti-monotonic upper bound f_n of f_s:

$$f_n(e') = \sum_{e\in\widetilde{E}(e',\epsilon_v,1,1)} \exp(\tau t_e), \quad (9)$$

where we only keep the node relaxation ratio ϵ_v while making the other two ratios maximally relaxed (i.e., with ratio 1), and relax the overlap ratio $\frac{|e'\cap e|}{|e|}$ to its upper bound of 1. We omit $\exp(\tau t_e)$ when timestamps are unavailable.

Theorem 1. *f_n is an anti-monotonic upper bound of f_s.*

Proof. **(Anti-monotonicity)** By definition, a function f_n is anti-monotonic if $e''\supseteq e'$ implies $f_n(e'')\le f_n(e')$. To show that f_n satisfies this property, it suffices to show that for any $e''\supseteq e'$, $\widetilde{E}(e'',\epsilon_v,1,1)\subseteq\widetilde{E}(e',\epsilon_v,1,1)$. Indeed, since $e''\supseteq e'$, the node-level condition imposed by e'' is at least as strict as that of e'. That is, if an observed hyperedge e satisfies $|\{\hat{e}\in\widetilde{E} : v'\notin\hat{e}\}|\le\epsilon_v\cdot|\widetilde{E}|$ for all $v'\in e''$, then the condition trivially holds for all $v'\in e'$, since $e'\subseteq e''$. Hence, any hyperedge satisfying the constraint for e'' also satisfies it for e', and the inclusion of support sets follows. Since f_n sums over these sets, we conclude $f_n(e')\ge f_n(e'')$.

(Upper bound) We aim to show that $f_s(e)\le f_n(e)$ for any candidate hyperedge e. This holds if $\widetilde{E}(e,\epsilon_v,\epsilon_e,\epsilon_t)\subseteq\widetilde{E}(e,\epsilon_v,1,1)$ since the overlap ratio in $f_s(e)$ is relaxed to its upper bound of 1 in $f_n(e)$. Indeed, the definition of $\widetilde{E}$ ensures that increasing the relaxation parameters ϵ_e and ϵ_t (i.e., making

the edge-level and time-level conditions less strict) can only expand the set of observed hyperedges that satisfy the similarity constraints. Therefore, any $e \in \widetilde{E}(e, \epsilon_v, \epsilon_e, \epsilon_t)$ also belongs to $\widetilde{E}(e, \epsilon_v, 1, 1)$, implying $\widetilde{E}(e, \epsilon_v, \epsilon_e, \epsilon_t) \subseteq \widetilde{E}(e, \epsilon_v, 1, 1)$. □

Computing f_n is NP-complete [22], and we further derive an upper bound f_t of f_n that allows efficient evaluation:

$$f_t(e') = \sum\nolimits_{e \in \widetilde{E}(e', 1, 1, \epsilon_v)} \exp(\tau t_e), \tag{10}$$

where we replace the node relaxation condition with the total relaxation condition with the same ratio. The key idea is that, when only total relaxation is involved, we can efficiently construct $\widetilde{E}(e', 1, 1, \epsilon_v)$ by greedily including hyperedges e in the ascending order w.r.t the number of missing nodes, i.e., $|v' \in e' : v' \notin e|$, until the relaxation ratio ϵ_v is reached.

Lemma 1. *f_t is an upper bound of f_s.*

Proof. It suffices to show that, $\widetilde{E}(e', \epsilon_v, 1, 1) \subseteq \widetilde{E}(e', 1, 1, \epsilon_v)$, for any e'. Indeed, for any observed hyperedge e such that $\forall v' \in e', |\{e \in \widetilde{E} : v' \notin e\}| \leq \epsilon_v |\widetilde{E}|$, we immediately have $|v' \in e', e \in \widetilde{E} : v' \notin e| = \sum_{v' \in e'} |\{e \in \widetilde{E} : v' \notin e\}| \leq \sum_{v' \in e'} \epsilon_v |\widetilde{E}| = \epsilon_v |e'||\widetilde{E}|$. □

Now, f_t can be used instead of f_n for pruning with improved efficiency, while maintaining theoretical guarantees. If a candidate e' satisfies $f_t(e') < \theta$ for some pruning threshold θ, then all supersets e'' of e' can be pruned without loss of optimality (cf. Eq. (9)):

$$f_s(e'') \overset{\text{upper bd.}}{\leq} f_n(e'') \overset{\text{anti-mono.}}{\leq} f_n(e') \overset{\text{upper bd.}}{\leq} f_t(e') < \theta. \tag{11}$$

In practice, we use f_t to quickly prune sub-optimal candidates, and then use the original scoring function f_s to evaluate the "surviving" candidates that pass the filtering of f_t.

D. DFS-based Candidate Exploration and Top-k Maintenance

We use a DFS-based search to explore the whole candidate set, and maintain a list of candidates with the highest scores on the fly. As mentioned in Sect. V-A, we first determine the target number k_i of hyperedges to predict for each size i, to match the original hyperedge size distribution in E.

We use DFS to explore the candidate set, from small candidates to large ones. Along the search, for each size i, we maintain a list E'_i of top-k_i candidates with size i and the corresponding threshold $\theta_i = \min_{e' \in E'_i} f_s(e')$. For each candidate e' with size $i = |e'|$, we use the efficient upper bound f_t to evaluate it. If $f_t(e') < \theta_i$, then we can safely skip e' and all its supersets (see Eq. (11)), without loss of optimality. Otherwise if e' passes the filtering of f_t (i.e., $f_t(e') \geq \theta_i$), we compute the original score $f_s(e')$ and update the top-k list and threshold accordingly.

After checking all candidates, we collect the top-k_i hyperedges from different hyperedge sizes i as the final predictions.

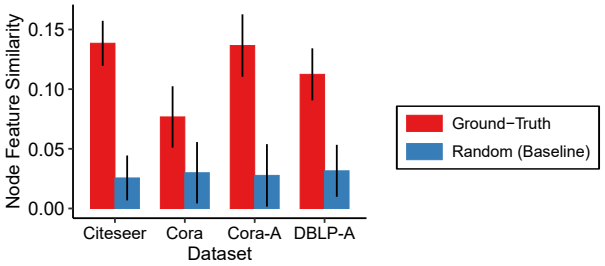


Fig. 2. **Nodes within a hyperedge have similar features.** Ground-truth hyperedges show higher Jaccard similarity between features of node pairs, suggesting new hyperedges tend to form between nodes with similar features.

E. Extension with Node Features

In the real world, many hypergraph datasets are accompanied by node features, and the proposed method HyperSearch can be easily extended to cases with node features. Specifically, each node v_i can be associated with a node-feature vector $x_i \in \mathbb{R}^f$ in dimension f. For the real-world datasets used in this work, the nodes in the Co-citation and Authorship datasets are publications (see Sect. III-B), and each node is associated with text information (the abstract of the corresponding publication). We extract one-hot bag-of-word features from such information as node features.

Observations. We analyze the similarity between node features within hyperedges, and have the following observations. For each hyperedge e, [6] we compute the Jaccard index between the node features of each node pair in e, take the average, and compare it to that in random hyperedges generated as in Sect. IV-A. As shown in Fig. 2, we observe that ground-truth hyperedges exhibit significantly higher node feature similarity between the node pairs than random ones. This suggests that new hyperedges are more likely to form among nodes with similar features.

Feature Weight. Based on our observations above, we introduce *feature weight* that prioritizes candidate hyperedges consisting of nodes with similar features. For each candidate e', we compute the average Jaccard index $X_{e'} = \frac{1}{\binom{|e'|}{2}} \sum_{v_i, v_j \in e'} \text{Jaccard}(x_i, x_j)$ between the node pairs in e', together with a tunable hyperparameter $\alpha \geq 0$ to adjust the significance of feature weight. With feature weight, the final scoring function is $\hat{f}_s(e') = (X_{e'})^\alpha f_s(e')$, and the corresponding upper bound for pruning is $\hat{f}_t(e') = (X_{e'})^\alpha f_t(e')$.

Discussion. A limitation regarding theoretical guarantees, when using node features, is that the average Jaccard index $X_{e'}$ is not anti-monotonic, and we thus cannot fully guarantee that the pruned candidates are never better than the kept ones. For theoretical rigor, one may still use f_t as the upper bound for pruning, since $\hat{f}_s(e'') \leq f_s(e'') \leq f_t(e')$, where we have used $(X_{e'})^\alpha \leq 1$. However, empirically we observe that using $\hat{f}_t(e')$ shows much better efficiency, which is likely because f_t is too loose as an upper bound compared to $\hat{f}_t$.

[6] Specifically, we use each hyperedge $e \in E_2$, following the setup in Sect. IV-A.

VI. EXPERIMENTS

In this section, we present experimental results to evaluate the proposed method HyperSearch, show its empirical superiority, and analyze the effectiveness of its key components. The experiments aim to answer the following research questions:

- **Q1. Accuracy**: How accurately does HyperSearch predict new hyperedges, compared to baselines?
- **Q2. Speed and Scalability**: How does the running time of HyperSearch compare to baselines, and how does it scale with data sizes and the number of hyperedges to predict?
- **Q3. Ablation Studies**: How does each component of HyperSearch contribute meaningfully to its prediction accuracy?
- **Q4. Hyperparameter Sensitivity**: How does the accuracy of HyperSearch vary with hyperparameter settings?

A. Experimental Settings

In this section, we present the dataset, method setups, and evaluation metrics. Detailed hardware and software specifications are provided in Sect. A of the online appendix [24].

1) Datasets: We used 10 real-world hypergraphs across five domains, as introduced in Sect. III-B (see Tab. I). For datasets with edge timestamps (i.e., the email, contact, and tags datasets), we split the datasets chronologically. Specifically, the earliest 80% of hyperedges were used as observed (training) hyperedges E_1, while the most recent 20% hyperedges were used as new (test) hyperedges E_2 we aim to predict. For the other datasets (i.e., the co-citation and authorship datasets), hyperedges were randomly split into 80% observed (training) hyperedges E_1 and 20% new (test) hyperedges E_2, using five different random seeds. For hyperparameter tuning, (the most recent or random) 20% of training hyperedges (i.e., 16% of the total) were set aside for validation.

For the splits, we make sure that the same hyperedge does not appear in both E_1 and E_2 (which is possible when repeated hyperedges exist). We also make sure that every node in E_2 also appears in E_1, i.e., there are no unseen nodes in E_2. The Cora, High, Primary, Ubuntu, and Math datasets contain only hyperedges of size up to 5. For computational efficiency, in the other datasets, only hyperedges up to size 10 were retained before splitting into E_1 and E_2. Additionally, hyperedges with rarely occurring hyperedge sizes ($< 1\%$) were removed. Details are provided in Sect. B of the online appendix [24].

2) Method Setups: All hyperparameters of HyperSearch (ϵ_v, ϵ_e, ϵ_t, τ, and α) were tuned based on validation performance for each dataset, and we used HyperSearch with the fine-tuned hyperparameters to predict new hyperedges in testing. Specifically, we tuned the hyperparameters through a grid search based on the performance w.r.t. Recall@1$\times$ (refer to Sect. VI-A3) on the validation set. We considered 27 combinations of the relaxation ratios (ϵ_v, ϵ_e, ϵ_t) where each ratio was set to one of $\{(\frac{1}{3}, \frac{1}{4}, \frac{1}{5})\}$ and additionally $\epsilon_v = \epsilon_e = \epsilon_t = 0$. In addition, the hyperparameters related to node feature similarity (α) and time weighting (τ) were each selected from the candidate set $\{0, 0.1, 1, 10\}$.

We evaluated competing methods, including (1) clique negative sampling (CNS), the strongest sampling-based method from [20], (2) HPRA [12], and (3) MHP [14]. The number of predicted hyperedges was set to $\{1, 2, 5\} \times |E_2|$ (see also Sect. VI-A3 for evaluation metrics).

- **CNS** [20]: It generates hyperedges by (1) randomly selecting an observed hyperedge e, (2) randomly choosing a node in $v \in e$, and (3) replacing v with another node that has common neighbors with all the other nodes in e.
- **HPRA** [12]: It generates hyperedge by (1) randomly selecting an initial node v_0, (2) iteratively adding nodes with high similarity scores with the already added nodes, based on resource allocation principles (e.g., based on the number of common neighbors).
- **MHP** [14]: It generates hyperedges by (1) starting from an empty query set, (2) selecting the first node via Preferential Attachment (i.e., high-degree nodes are likely to be chosen), and (3) iteratively adding nodes based on probabilities predicted by a trained model.

Note that CNS and HPRA have no hyperparameters to tune. S ee Sect. II for more details and discussions of these baselines.

For the baselines that do not use Hypergraph Neural Networks (HNNs), i.e., CNS and HPRA, we further extended them by applying HNNs as a post-processing step to refine hyperedge candidates generated by them. We used different HNN models: MHP [14], AHP [15], Hyper-SAGNN [13], and NHP [16]. For each HNN model, the suffix "-C" was added when the model was applied to CNS, and the suffix "-H" was added when the model was applied to HPRA. For example, MHP-C refers to the approach where MHP is applied to refine hyperedge candidates generated by CNS, while AHP-H indicates that AHP is used to filter hyperedges predicted by HPRA. In these extended two-stage models, all HNN models were initially trained for the hyperedge prediction task using their respective loss functions in their original implementation. In Sect. C of the online appendix [24], we provide a summarizes the hyperparameter search spaces used for the HNN models employed in our experiments.

After training, they were applied as a post-processing step to rank and refine the hyperedge candidates generated in the first stage. Specifically, we first used CNS or HPRA to generate twice the target number (i.e., $2k$) of predicted hyperedges, and then used an HNN model to select the top-k ones.

3) Evaluation Metrics: To evaluate prediction performance, we used Recall@$\mathcal{K}$, a widely adopted metric in (hyper)edge prediction tasks. Recall@$\mathcal{K}$ measures the proportion of ground-truth new hyperedges that are correctly included in the predicted set when $\mathcal{K}$ hyperedges are predicted. In this study, the parameter $\mathcal{K}$ is set to represent a multiple of the number of hyperedges in E_2. For instance, if $\mathcal{K} = 2\times$, the number of predicted hyperedges is $2 \times |E_2|$.

In addition to Recall@$\mathcal{K}$, we also evaluated prediction accuracy using the average F1 score [25], which considers the similarity between predicted and ground-truth hyperedges. The formal definition and the experimental results using the metric are provided in Sect. D of the online appendix [24].

TABLE II

Q1. ACCURACY. HYPERSEARCH PERFORMS BETTER THAN ALL THE BASELINES CONSISTENTLY ACROSS ALL SETTINGS. WE REPORT THE AVERAGE VALUES AND STANDARD DEVIATIONS OVER THE FIVE RANDOM SPLITS OF RECALL@$\mathcal{K}$, ACROSS THE FOUR DATASETS WITHOUT EDGE TIMESTAMPS. FOR EACH SETTING, THE BEST AND SECOND-BEST METHODS ARE HIGHLIGHTED IN GREEN AND YELLOW, RESPECTIVELY.

Dataset	**Citeseer**			**Cora**			**Cora-A**			**DBLP-A**		
Method (↓) / $\mathcal{K}$ (→)	1×	2×	5×	1×	2×	5×	1×	2×	5×	1×	2×	5×
HyperSearch (Proposed)	8.2 (1.6)	10.9 (1.5	17.9 (1.8)	7.5 (1.8)	10.0 (2.0)	14.6 (1.5)	7.3 (3.6)	10.9 (2.5)	16.4 (2.9)	5.4 (0.1)	8.4 (0.2)	14.3 (0.4)
CNS	1.5 (0.2)	3.3 (0.8)	8.8 (1.4)	2.9 (2.1)	5.9 (1.5)	12.5 (2.1)	0.3 (0.2)	0.6 (0.6)	2.1 (0.8)	0.7 (0.2)	1.2 (0.1)	2.7 (0.2)
HPRA	0.2 (0.4)	0.3 (0.4)	0.8 (0.6)	0.2 (0.2)	0.6 (0.5)	2.3 (1.5)	0.0 (0.0)	0.1 (0.2)	0.1 (0.2)	0.0 (0.0)	0.0 (0.0)	0.1 (0.0)
MHP	2.8 (1.1)	4.4 (1.3)	8.9 (1.4)	1.2 (0.9)	2.4 (1.1)	6.0 (1.6)	0.8 (0.2)	1.6 (0.2)	6.1 (2.8)	-	-	-
MHP-C	2.3 (1.0)	5.7 (1.7)	-	4.2 (1.3)	8.0 (1.5)	-	0.4 (0.4)	1.4 (0.7)	2.6 (0.5)	-	-	-
AHP-C	2.4 (0.9)	5.2 (1.2)	-	4.0 (1.0)	8.5 (1.8)	-	0.4 (0.4)	0.9 (0.6)	1.7 (0.7)	-	-	-
SAGNN-C	1.8 (0.6)	4.3 (1.4)	-	3.8 (1.7)	7.5 (2.2)	-	0.3 (0.3)	0.7 (0.5)	1.5 (0.6)	0.7 (0.1)	1.2 (0.2)	2.3 (0.4)
NHP-C	2.3 (0.9)	5.5 (1.2)	-	4.2 (1.3)	7.4 (1.2)	-	0.4 (0.3)	0.9 (0.3)	2.2 (0.6)	0.9 (0.2)	1.6 (0.2)	3.4 (0.2)
MHP-H	0.3 (0.4)	0.7 (0.6)	-	0.6 (0.5)	1.9 (1.2)	3.4 (1.4)	0.1 (0.1)	0.1 (0.1)	0.1 (0.1)	-	-	-
AHP-H	0.0 (0.0)	0.1 (0.1)	-	0.5 (0.0)	1.4 (0.0)	1.8 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	-	-	-
SAGNN-H	0.2 (0.2)	0.4 (0.3)	-	0.4 (0.4)	1.2 (0.8)	2.1 (1.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	-
NHP-H	0.1 (0.2)	0.3 (0.3)	-	0.6 (0.5)	1.9 (1.2)	3.4 (1.5)	0.1 (0.2)	0.1 (0.2)	0.1 (0.2)	0.0 (0.0)	0.0 (0.0)	-

-: out-of-time (> 2 days).

TABLE III

Q1. ACCURACY. HYPERSEARCH PERFORMS BETTER THAN ALL THE BASELINES IN MOST SETTINGS. WE REPORT THE RECALL@$\mathcal{K}$ ACROSS THE SIX DATASETS WITH EDGE TIMESTAMPS (NO STANDARD DEVIATION SINCE THERE IS ONLY ONE CHRONOLOGICAL SPLIT). FOR EACH SETTING, THE BEST AND SECOND-BEST METHODS ARE HIGHLIGHTED IN GREEN AND YELLOW, RESPECTIVELY.

Dataset	**Enron**			**Eu**			**High**			**Primary**			**Ubuntu**			**Math-sx**		
Method (↓) / $\mathcal{K}$ (→)	1×	2×	5×	1×	2×	5×	1×	2×	5×	1×	2×	5×	1×	2×	5×	1×	2×	5×
HyperSearch (Proposed)	16.1	25.6	33.1	12.4	17.3	26.8	14.8	18.3	27.3	7.3	11.8	20.8	12.0	15.4	20.6	12.1	17.3	24.5
CNS	10.3	16.4	29.7	5.1	10.9	22.0	12.6	13.8	18.1	4.5	7.3	11.9	1.6	2.9	6.7	3.4	6.0	11.7
HPRA	1.7	5.8	9.2	3.5	5.8	10.2	9.0	14.8	28.1	4.7	8.1	20.4	1.1	2.0	4.4	2.2	3.9	8.1
MHP	0.3	0.6	3.6	0.3	1.0	3.4	0.9	2.9	7.4	4.3	7.7	21.1	-	-	-	-	-	-
MHP-C	7.6	14.9	22.0	7.4	14.1	22.7	4.3	5.7	8.3	4.1	5.7	9.6	-	-	-	-	-	-
SAGNN-C	6.0	8.2	14.6	8.6	16.0	24.5	7.2	8.5	9.9	5.1	7.9	11.9	2.3	4.3	8.7	4.7	7.9	14.5
NHP-C	13.3	20.3	29.8	8.1	14.9	23.3	9.7	11.6	13.7	5.3	8.5	13.1	1.8	3.5	7.1	3.6	6.1	11.2
MHP-H	4.6	5.4	9.5	4.4	6.5	14.2	9.1	16.3	31.8	5.4	11.4	22.6	-	-	-	-	-	-
SAGNN-H	2.8	3.4	7.1	5.4	8.1	17.8	10.2	18.1	34.1	4.5	9.4	19.5	2.0	3.5	-	3.7	6.5	-
NHP-H	4.5	5.2	9.5	4.6	6.7	14.2	12.4	16.6	32.7	6.0	11.7	22.5	1.6	2.8	-	2.6	4.6	-

-: out-of-time (> 2 days).

B. Q1. Accuracy (Tables II & III)

We evaluated the accuracy of HyperSearch and baselines, and the results are presented in Tables II and III.

Notably, HyperSearch consistently outperformed all baselines, including deep learning-based models and rule-based approaches (e.g., CNS and HPRA), across most datasets and experimental settings. The only exceptions were in Recall@5× on the Contact domain datasets (High and Primary), where HyperSearch did not achieve the highest performance. Based on our analysis, this is likely because for those datasets, not enough promising (e.g., those with high overlap with the observed ones) candidates are available. A detailed analysis of these cases is provided in Sect. E of the online appendix [24].

It is also worth noting that HPRA exhibited scalability limitations, particularly in datasets where the observed hyperedges are divided into multiple connected components (see Sect. I), performing worse than even simple negative sampling methods (e.g., CNS). In contrast, HyperSearch guaranteed flexible exploration of all promising candidates, regardless of the presence of disconnected components.

As shown in Sect. D of the online appendix [24], the results were consistent in terms of average F1 score.

Case Studies. To qualitatively evaluate the semantic coherence of nodes within predicted hyperedges, we conducted case studies on the two tag datasets, Math and Ubuntu, where the identities of the nodes (tags) are available. For each hyperedge size from 2 to 5, we selected the top-scoring predicted hyperedge generated by HyperSearch, which are:

- **Math examples**: [ring-theory, noetherian], [matrices, vectors, vector-spaces], [group-theory, finite-groups, field-theory, abstract-algebra], [calculus, sequences-and-series, real-analysis, integration, convergence]
- **Ubuntu examples**: [drivers, xorg], [boot, grub2, btrfs], [dual-boot, boot, live-usb, grub2], [partitioning, grub2, 16.04, dual-boot, boot]

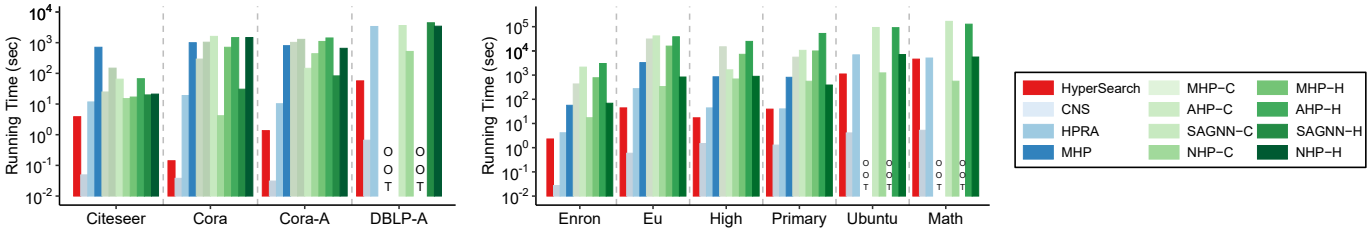
The results show that HyperSearch effectively identifies groups (hyperedges) of semantically related tags (nodes).

C. Q2. Speed and Scalability (Figs. 3 & 5)

We evaluated the runtime of the proposed method HyperSearch and baselines, and analyzed how HyperSearch scales with the size of the input hypergraph. The results, presented in Fig. 3 and Fig. 5, reveal the following key observations:

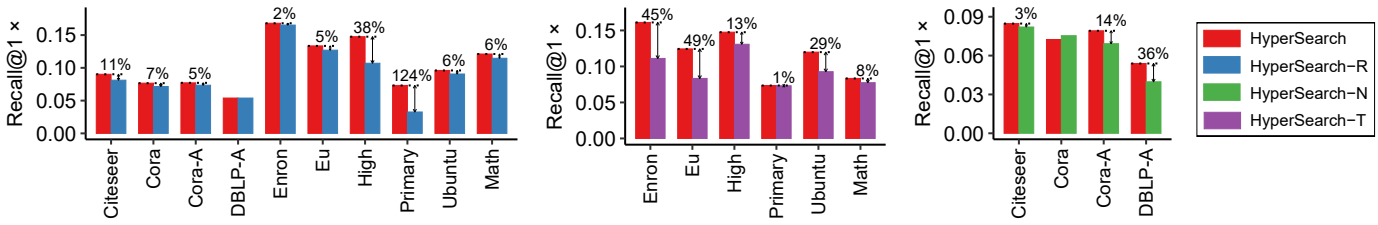
- HyperSearch runs faster than deep learning-based methods (highlighted in green in Fig. 3) in most cases, demonstrating its superior computational efficiency across diverse datasets.
- The runtime of HyperSearch scales almost linearly with the number of hyperedges in the input hypergraph, with a regression slope of 1.05 (Fig. 5). This suggests that HyperSearch maintains scalability as the data size increases.

In Sect. G of the online appendix [24], we show that this scalability persisted on larger synthetic hypergraphs, generated by replicating each original hypergraph up to five times.



(a) Without edge timestamps (b) With edge timestamps

Fig. 3. **Q2. Runtime comparison.** HyperSearch achieves lower runtime than deep learning-based methods (shown in green) in most cases. For each dataset, we report the running time of HyperSearch and baselines.



(a) **Relaxed overlap count (Sect. V-B1)** (b) **Time weight (Sect. V-B2)** (c) **Feature weight (Sect. V-E)**

Fig. 4. **Q3. Ablation studies.** Each component of HyperSearch meaningfully contributes to its predictive performance. HyperSearch consistently achieves superior or comparable accuracy compared to its variants with some component missing.

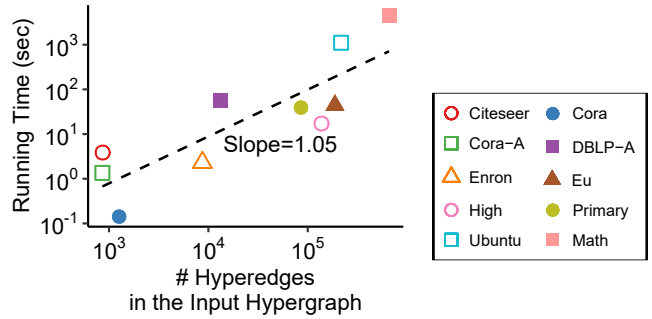


Fig. 5. **Q2. Scalability of HyperSearch.** The runtime of HyperSearch scales nearly linearly with the input hypergraph size, with a regression slope of 1.05, indicating strong scalability. In the plot, we report the number of hyperedges and the running time of HyperSearch on each dataset

D. Q3. Ablation Studies (Fig. 4)

We conducted ablation studies on (1) relaxed overlap count in Sect. V-B1, (2) time weight in Sect. V-B2, and (3) feature weight in Sect. V-E to assess their impact on performance. Specifically, we compared Recall@1× between HyperSearch and its variants without each component: (1) HyperSearch-R without relaxation (i.e., $\epsilon_v = \epsilon_e = \epsilon_t = 0$), (2) HyperSearch-T without time weight (i.e., $\tau = 0$) , and (3) HyperSearch-N without feature weight (i.e., $\alpha = 0$). As shown in Fig. 4, HyperSearch consistently achieves superior or comparable performance compared to its variants, demonstrating that each component meaningfully contributes to its prediction accuracy.

E. Q4. Hyperparameter Sensitivity (Fig. 6)

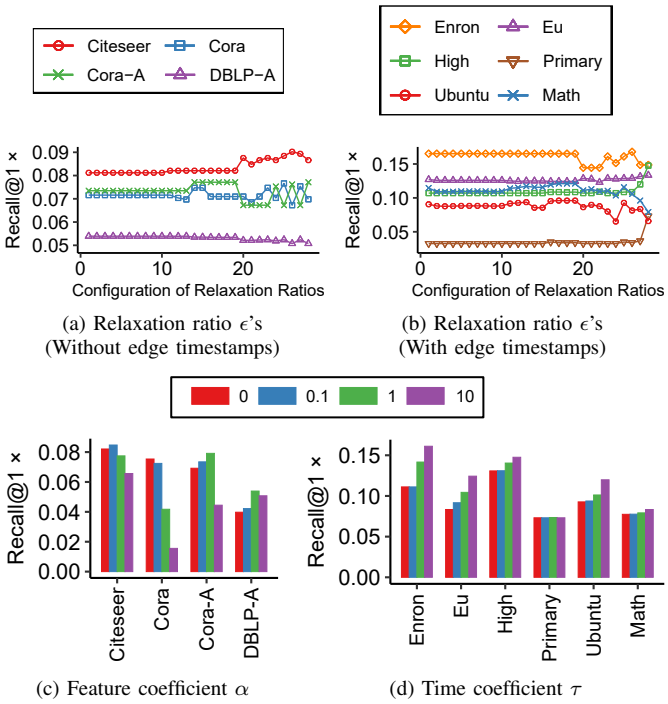
We analyzed the sensitivity of all hyperparameters in HyperSearch, including the relaxation ratios (ϵ_v, ϵ_e, ϵ_t), the feature weight (α), and the time weight (τ). Specifically, we evaluated Recall@1× by varying (ϵ_v, ϵ_e, ϵ_t) across 28 combinations described in Sect. VI-A2 and by varying α and τ in $\{0, 0.1, 1, 10\}$, for each dataset.

As shown in Figs. 6(a) and (b), the optimal relaxation ratios (ϵ_v, ϵ_e, ϵ_t) vary across datasets. Fig. 6(c) shows that, in most cases, HyperSearch achieves peak accuracy at specific $\alpha \in \{0.1, 1\}$ for most datasets. On the other hand, as shown in Fig. 6(d), increasing $\tau \in \{0, 0.1, 1, 10\}$ monotonically improves the accuracy of HyperSearch in 5 out of 6 datasets. These results suggest that the impact of hyperparameters depends on dataset characteristics, as different datasets benefit from varying degrees of relaxation, time weight, and feature weight. Notably, in our experiments, via validation, we found hyperparameters that gave competitive performance, making HyperSearch outperform baselines in most cases.

F. Additional Experiments

Due to the page limit, we provide extra experimental results in the online appendix [24], summarized as follows:

- **Combination with neural networks (Sect. F):** We tried applying a filtering step using hypergraph neural networks to refine the predictions of HyperSearch, similar to what we have done for CNS and HPRA (see Sect. VI-A2). However, this combination did not consistently improve the performance of HyperSearch, indicating that its score function based on the patterns in real-world hypergraphs (see Sect. IV) is already effective.
- **Scalability w.r.t. the number of predictions (Sect. H):** The running time of HyperSearch scaled nearly linearly with the number of predictions it produced.
- **Failure analysis (Sect. E):** We investigated potential reasons for the relatively poor performance of HyperSearch on the Contact domain datasets.
- **Additional metric (Sect. D) and large-scale synthetic datasets (Sect. G)**: We confirmed that our results remained consistent in terms of average F1 score, and across large-scale synthetic datasets created by replicating hypergraphs.



(a) Relaxation ratio ϵ's (Without edge timestamps)

(b) Relaxation ratio ϵ's (With edge timestamps)

(c) Feature coefficient α

(d) Time coefficient τ

Fig. 6. **Q4. Hyperparameter sensitivity.** The best hyperparameter choices vary from dataset to dataset. The trends of the accuracy of HyperSearch across different hyperparameters (ϵ_v, ϵ_e, ϵ_t), α, and τ.

VII. Conclusion and Discussion

In this work, we studied the problem of hyperedge prediction. We identified two key limitations of existing methods (Sect. I). We analyzed and discussed several observations in real-world hypergraphs on structure and timestamps (Sect. IV). We proposed a novel method, HyperSearch, for hyperedge prediction, consisting of two key components: (1) a scoring function based on our observations to evaluate candidate hyperedges, and (2) an efficient search scheme using an anti-monotonic upper bound of the scoring function (Sect. V). We conducted extensive experiments on 10 real-world hypergraphs across five domains to validate the empirical superiority of HyperSearch regarding both accuracy and efficiency (Sect. VI).

Discussion on Limitations. While HyperSearch demonstrates strong empirical efficiency, its theoretical complexity is high due to reliance on integer programming [22]. Moreover, for hypergraphs with features, incorporating the average Jaccard index compromises the anti-monotonicity, as discussed in Sect. V-E, where we discussed a workaround for theoretical rigor (i.e., use 1 as a trivial upper bound of the average Jaccard index). We would like to explore theoretically sound and empirically effective upper bounds when additional information (e.g., node features) is available, as a future direction.

Reproducibility. Our code, datasets, and appendix are publicly available at https://github.com/jin-choo/HyperSearch/.

Acknowledgment

This work was partly supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2022-II220157, Robust, Fair, Extensible Data-Centric Continual Learning) (No. RS-2024-00457882, AI Research Hub Project) (RS-2019-II190075, Artificial Intelligence Graduate School Program (KAIST)). This research was partly supported by National Research Foundation of Korea (NRF) (RS-2023-00209473), Korea Basic Science Institute (National research Facilities and Equipment Center) grant funded by the Ministry of Science and ICT (RS-2024-00401676).

References

[1] A. R. Benson, R. Abebe, M. T. Schaub, A. Jadbabaie, and J. Kleinberg, "Simplicial closure and higher-order link prediction," *PNAS*, vol. 115, no. 48, pp. E11 221–E11 230, 2018.

[2] F. Klimm, C. M. Deane, and G. Reinert, "Hypergraphs for predicting essential genes using multiprotein complex data," *Journal of Complex Networks*, vol. 9, no. 2, p. cnaa028, 2021.

[3] I. Iacopini, G. Petri, A. Baronchelli, and A. Barrat, "Group interactions modulate critical mass dynamics in social convention," *Communications Physics*, vol. 5, no. 1, p. 64, 2022.

[4] G. Lee, F. Bu, T. Eliassi-Rad, and K. Shin, "A survey on hypergraph mining: Patterns, tools, and generators," *ACM Computing Surveys*, vol. 57, no. 8, pp. 1–36, 2025.

[5] J. L. Juul, A. R. Benson, and J. Kleinberg, "Hypergraph patterns and collaboration structure," *Frontiers in Physics*, vol. 11, p. 1301994, 2024.

[6] D. Liben-Nowell and J. Kleinberg, "The link prediction problem for social networks," in *CIKM*, 2003.

[7] P. Wang, B. Xu, Y. Wu, and X. Zhou, "Link prediction in social networks: the state-of-the-art," *Science China Information Sciences*, vol. 1, no. 58, pp. 1–38, 2015.

[8] X. Wang and G. Sukthankar, "Link prediction in heterogeneous collaboration networks," *Social network analysis-community detection and evolution*, pp. 165–192, 2014.

[9] D. Lande, M. Fu, W. Guo, I. Balagura, I. Gorbov, and H. Yang, "Link prediction of scientific collaboration networks based on information retrieval," *WWW*, 2020.

[10] S. Jin, Y. Hong, L. Zeng, Y. Jiang, Y. Lin, L. Wei, Z. Yu, X. Zeng, and X. Liu, "A general hypergraph learning algorithm for drug multi-task predictions in micro-to-macro biomedical networks," *PLOS Computational Biology*, vol. 19, no. 11, p. e1011597, 2023.

[11] K. M. Saifuddin, B. Bumgardner, F. Tanvir, and E. Akbas, "Hygnn: Drug-drug interaction prediction via hypergraph neural network," in *ICDE*, 2023.

[12] T. Kumar, K. Darwin, S. Parthasarathy, and B. Ravindran, "Hpra: Hyperedge prediction using resource allocation," in *WebSci*, 2020.

[13] R. Zhang, Y. Zou, and J. Ma, "Hyper-sagnn: a self-attention based graph neural network for hypergraphs," in *ICLR*, 2020.

[14] T. Yu, S. Y. Lee, H. Hwang, and K. Shin, "Prediction is not classification: On formulation and evaluation of hyperedge prediction," in *ICDMW*, 2024.

[15] H. Hwang, S. Lee, C. Park, and K. Shin, "Ahp: Learning to negative sample for hyperedge prediction," in *SIGIR*, 2022.

[16] N. Yadati, V. Nitin, M. Nimishakavi, P. Yadav, A. Louis, and P. Talukdar, "Nhp: Neural hypergraph link prediction," in *CIKM*, 2020.

[17] M. Zhang, Z. Cui, S. Jiang, and Y. Chen, "Beyond link prediction: Predicting hyperlinks in adjacency space," in *AAAI*, 2018.

[18] S.-e. Yoon, H. Song, K. Shin, and Y. Yi, "How much and when do we need higher-order information in hypergraphs? a case study on hyperedge prediction," in *WWW*, 2020.

[19] K. Tu, P. Cui, X. Wang, F. Wang, and W. Zhu, "Structural deep embedding for hyper-networks," in *AAAI*, 2018.

[20] P. Patil, G. Sharma, and M. N. Murty, "Negative sampling for hyperlink prediction in networks," in *PAKDD*, 2020.

[21] F. Chung and L. Lu, "The average distances in random graphs with given expected degrees," *PNAS*, vol. 99, no. 25, pp. 15 879–15 882, 2002.

[22] A. K. Poernomo and V. Gopalkrishnan, "Towards efficient mining of proportional fault-tolerant frequent itemsets," in *KDD*, 2009.

[23] T. Achterberg, "Scip: solving constraint integer programs," *Mathematical Programming Computation*, vol. 1, pp. 1–41, 2009.

[24] H. Choo, F. Bu, H. Hwang, Y.-G. Yoon, and K. Shin, "Hypersearch: Online appendix," https://github.com/jin-choo/HyperSearch/blob/master/appendix.pdf, 2025.

[25] J. Yang and J. Leskovec, "Overlapping community detection at scale: a nonnegative matrix factorization approach," in *WSDM*, 2013.