Convolutional Signal Propagation: A Simple Scalable Algorithm for Hypergraphs

Pavel Procházka[1], Marek Dědič[12] (✉)[0000−0003−1021−8428], and Lukáš Bajer[1][0000−0002−9402−6417]

[1] Cisco Systems, Inc., Karlovo náměstí 10, Prague, 120 00, Czech Republic
{paprocha,madedic,lubajer}@cisco.com
[2] Faculty of Nuclear Sciences and Physical Engineering, Czech Technical University in Prague, Břehová 7, Prague, 110 00, Czech Republic

Abstract Last decade has seen the emergence of numerous methods for learning on graphs, particularly Graph Neural Networks (GNNs). These methods, however, are often not directly applicable to more complex structures like bipartite graphs (equivalent to hypergraphs), which represent interactions among two entity types (e.g. a user liking a movie). This paper proposes Convolutional Signal Propagation (CSP), a non-parametric simple and scalable method that natively operates on bipartite graphs (hypergraphs) and can be implemented with just a few lines of code. After defining CSP, we demonstrate its relationship with well-established methods like label propagation, Naive Bayes, and Hypergraph Convolutional Networks. We evaluate CSP against several reference methods on real-world datasets from multiple domains, focusing on retrieval and classification tasks. Our results show that CSP offers competitive performance while maintaining low computational complexity, making it an ideal first choice as a baseline for hypergraph node classification and retrieval. Moreover, despite operating on hypergraphs, CSP achieves good results in tasks typically not associated with hypergraphs, such as natural language processing.

Keywords: Hypergraph representation learning · Hypergraph convolution · Label Propagation · Model complexity · Naive Bayes

1 Introduction

In the modern world, an overwhelming amount of data has an internal structure, oftentimes forming complex networks that can be represented as graphs. Efficiently mining information from this data is crucial for a wide range of applications spanning various domains such as social networks, biology, physics or cybersecurity. Graph Neural Networks (GNNs) have emerged as the dominant tool for handling such data due to their ability to leverage the graph structure for predictive and analytical tasks. However, despite their success, GNNs come with notable challenges, including high computational complexity during training, numerous hyperparameters that require fine-tuning, lack of straightforward interpretability, and the necessity of dedicated computational infrastructure such as GPUs.

Given these limitations, baseline algorithms play a vital role as complementary tools to GNNs. These baselines, often much less complex, provide an efficient way of generating preliminary results. In many cases, these simpler methods are even sufficiently effective to be used as-is for the problem at hand.

Section 2 introduces the problem being solved in this work and Section 3 provides an overview of other related works. Section 4 is the most substantive part of this paper, introducing the CSP

method (Section 4.1). In Section 4.5, we show CSP to be a straightforward extension of the well-known label propagation method to hypergraphs. Also provided is a comparison of CSP to other methods, such as hypergraph convolutional networks (Section 4.4) and the naive Bayes classifier (Section 4.6). Finally, Section 5 provides an experimental evaluation and comparison of CSP and several alternative methods.

2 Problem Statement

Assume that we have structured data with relationships that can be translated into a hypergraph or a bipartite graph. These structures can represent various scenarios such as users rating movies, users accessing web domains, emails containing attachments, authors writing papers, papers being co-cited, and tokens being contained in texts. Some data might also come with constraints such as large volume or privacy restrictions, like those found in emails. These structures inherently provide a relationship between entities, enabling many applications to leverage these relationships, such as movie recommendations, mining malicious web content (like emails or domains), or text classification.

Our goal is to find a flexible method that can be applied to tasks involving structured data. This flexibility is sought in terms of performance, numerical complexity, and adaptability. We aim to develop a method that can efficiently handle and extract meaningful information from these complex relationships. Whether we are looking to extract interesting entities, which aligns with a retrieval scenario, or view the task as a simple classification problem, the method should be versatile enough to adapt to these needs.

We can formalize this problem using either a bipartite graph or a hypergraph. By representing the data in these graph structures, we can better understand and utilize the inherent relationships between entities. This formalization allows for the application of graph-based algorithms, which can improve the effectiveness of tasks like classification and retrieval.

2.1 Notations and Definitions

We consider a finite set of items of interest $V = \{v_1, \ldots, v_n\}$, referred to as *nodes*. A family of subsets of V denoted by $E = \{e_1, \ldots, e_m\} \subseteq 2^V$ is referred to as *hyperedges*. The nodes and hyperedges togerher form a *hypergraph* $\mathcal{H} = (V, E)$. The structure of the hypergraph can also be described by an *incidence matrix* $\mathbf{H} \in \{0,1\}^{n \times m}$, where $\mathbf{H}_{i,j} = 1$ if $v_i \in e_j$, and 0 otherwise. Every hypergraph can alternatively be described by a bipartite *incidence graph*, also called the Levi graph [22]. This bipartite graph $\mathcal{G}_{\text{bip}} = (V \cup E, E_{\text{bip}})$ has as its two partitions the nodes and hyperedges of $\mathcal{H}$, and its edges represent a node in V belonging to an edge in E, formally $E_{\text{bip}} = \{(v_i, e_j) \in V \times E | v_i \in e_j\}$.

The degree $\mathrm{d}(v)$ of node v is defined as the number of edges that contain the node, that is $\mathrm{d}(v) = |\{e_i \in E | v \in e_i\}|$. Similarly, the degree $\delta(e)$ of the edge e is defined as the number of nodes it contains, i.e. $\delta(e) = |e|$. We also establish a diagonal node-degree matrix $\mathbf{D}_v \in \mathbb{N}^{n \times n}$ with $(\mathbf{D}_v)_{i,i} = \mathrm{d}(v_i)$ and $(\mathbf{D}_v)_{i,j} = 0$ for $i \neq j$. Analogously, the hyperedge-degree matrix is a diagonal matrix $\mathbf{D}_e \in \mathbb{N}^{m \times m}$ with $(\mathbf{D}_e)_{i,i} = \delta(e_i)$ and $(\mathbf{D}_e)_{i,j} = 0$ for $i \neq j$.

We consider for each node in the hypergraph some kind of signal that is to be propagated through the hyperedges. Let the signal be a d-dimensional vector $\boldsymbol{x}_i$ for each node, giving for the whole hypergraph a matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$. In the following parts of this paper, we will explore several ways of defining such a signal, with an overview provided in Section 4.3.

Within this work, we are interested in two transductive tasks on hypergraphs: classification on V and retrieval of positive nodes from V. Both tasks assume a training set of nodes $V_{\text{train}} \subset V$ where the labels of nodes are known. In case of classification, the goal is to predict the label for all nodes in V. The retrieval task aims to sort the nodes in the testing set $V \setminus V_{\text{train}}$ such that the number of positive nodes in top K positions is maximized.

3 Related work

Mining information from structured (graph-like) data is one of the central problems in machine learning. The most straightforward way to handle this is to translate the structure into features and apply traditional machine learning techniques, such as logistic regression, random forests, and naive Bayes, to these features. Naive Bayes [26], in particular, provides a bridge to a second large family of learning methods on graphs: Bayesian methods, where the graph forms a structure for modelling random variables. A critical problem associated with Bayesian methods is inference, which is often intractable. The sum product algorithm [18] combats the tractability of a system by its decomposition to a product of local functions. One instance of this framework is probabilistic threat propagation [4], which is used for spreading information about malicious actors through the network.

The translation of structured data into features is also a non-trivial problem. While some methods can handle sparse, high-dimensional feature vectors, the majority cannot. Several methods are suited for finding low-dimensional representations of structured data. For example, non-negative matrix factorization [20] (NMF) decomposes a large sparse matrix into the product of two low-dimensional matrices. Node2vec [10] applies contrastive learning to find node representations where neighboring nodes are close in feature space. Spectral positional encodings [6] provide representations given by eigenvectors of the Laplacian matrix, and distance encodings [23,2] offer global information about where a node is located in the graph. However, except for NMF, these methods are suited for graphs and cannot be directly applied to hypergraphs.

There are many papers on learning algorithms for graphs, such as GraphSAGE [11], graph convolutional networks [16], and graph attention networks [32]. Nevertheless, their application to hypergraphs is not straightforward. The origins of learning transductive tasks stretch back to the seminal work [34]. More recently, Hypergraph neural networks [7], Dynamic HGNNs [15], and HyperGCN [33] build upon the convolutional learning schema introduced in [17] while works such as [1] aim to bring both convolutional as well as attention to the context of hypergraphs. The proposed method can also be viewed as an extension of label propagation [35,14] or feature propagation [31]. While there do exists algorithm for label propagation in hypergraphs [13,21], the proposed method aims to be comparatively simpler to understand, implement and calculate. The proposed method is an extension of our previous work in the specific domain of computer network security [30].

4 Convolutional Signal Propagation

We present Convolutional Signal Propagation (CSP), a method for signal propagation on hypergraphs. In the following subsections, CSP is first introduced in the general setting, followed by a comparison to established approaches and a discussion of possible variants inspired by them. Finally, applications of CSP to different kinds of signals in hypergraph tasks are discussed.

4.1 Method overview

The proposed algorithm propagates a node signal $\mathbf{X}$ (see Section 4.3 for a discussion of possible signal types) through the hypergraph $\mathcal{H}$. The basic version of CSP consists in a simple averaging of $\mathbf{X}$ across the hyperedges and nodes of the graph. This averaging can be repeated to obtain smoother final representations, resulting in a multi-step CSP process generating a sequence of representations $\mathbf{X}^{(l)}$, where $\mathbf{X}^{(0)} = \mathbf{X}$.

In each step, the representation $\mathbf{X}^{(l)}$ of the nodes is first propagated to the hyperedges to obtain their representations

$$\boldsymbol{r}_j^{(l)} = \frac{1}{\delta\left(e_j\right)} \sum_{\substack{i \\ v_i \in e_j}} \boldsymbol{x}_i^{(l)} \tag{1}$$

that is the average of the representation of the individual nodes contained in the hyperedge. In the second step, this hyperedge representation is propagated again into nodes:

$$\boldsymbol{x}_k^{(l+1)} = \frac{1}{\mathrm{d}\left(v_k\right)} \sum_{\substack{j \\ v_k \in e_j}} \boldsymbol{r}_j^{(l)}. \tag{2}$$

The steps 1 and 2 constitute the proposed Convolutional Signal Propagation algorithm, which can be summarily written as

$$\boldsymbol{x}_k^{(l+1)} = \frac{1}{\mathrm{d}\left(v_k\right)} \sum_{\substack{j \\ v_k \in e_j}} \frac{1}{\delta\left(e_j\right)} \sum_{\substack{i \\ v_i \in e_j}} \boldsymbol{x}_i^{(l)}. \tag{3}$$

Using notation established in Section 2.1, Equation 3 can be rewritten into the matrix form

$$\mathbf{X}^{(l+1)} = \mathbf{D}_v^{-1}\mathbf{H}\mathbf{D}_e^{-1}\mathbf{H}^T\mathbf{X}^{(l)}. \tag{4}$$

Equation 4 describes a basic variant of the proposed algorithm. In Section 4.7, various modifications of CSP are discussed.

4.2 On Efficient Implementation of Convolutional Signal Propagation

While Equation 4 shows an efficient way of mathematically expressing the CSP algorithm, the algorithm itself is also efficient when it comes to its implementation and computational complexity. Algorithm 1 shows an implementation of CSP in a single SQL query and Algorithm 2 shows a simple implementation in Python using the Pandas [24] library. These implementations essentially materialize Equations 1 and 2.

4.3 Application of CSP to different signals in hypergraphs

The construction of Convolutional Signal Propagation in Section 4.1 was a general one, assuming a signal matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$. In practice, one can use CSP to propagate various kinds of "signals" in the hypergraph. Namely, the matrix $\mathbf{X}$ may represent actual features as provided in the underlying graph dataset. This setting leads to a method similar to feature propagation [31] or hypergraph convolution [7]. Such an approach is elaborated further in Section 4.4. Alternatively, an approach

Algorithm 1 An SQL implementation of CSP (3), where graph is a table with columns nodeId, edgeId and property. While nodeId and edgeId desribe the hypergraph, property defines the task.

```sql
SELECT nodeId, AVG(edgeProperty) AS updatedProperty
FROM (
    SELECT nodeId, AVG(property) OVER (PARTITION BY edgeId) AS edgeProperty
    FROM graph
)
GROUP BY nodeId;
```

Algorithm 2 A Pandas implementation of CSP (3), where df is a DataFrame with columns nodeId, edgeId and property. While nodeId and edgeId desribe the hypergraph, property defines the task.

```python
import pandas as pd
def CSP_layer(df: pd.DataFrame) -> pd.DataFrame:
    return df.assign(edgeProperty=df.groupby('edgeId')['property'].transform('mean')) \
        .groupby('nodeId')['edgeProperty'].mean() \
        .reset_index().rename(columns={'edgeProperty': 'updatedProperty'})
```

based on label propagation [35] may be obtained by taking as $\mathbf{X}$ a version of the label matrix $\mathbf{Y}$ masked by the training set, a setting described in Section 4.5. Finally, the hyperedges of $\mathcal{H}$ may be used as the signal, essentially representing indicators of node similarity based on their presence in a common hyperedge. Such an approach corresponds to setting $\mathbf{X} = \mathbf{H}$ and is described in Section 4.6.

4.4 Comparison with Hypergraph Convolution

A single layer of the Hyper-Conv hypergraph neural network by [1] is defined as

$$\mathbf{X}^{(l+1)} = \sigma(\mathbf{D}_v^{-1}\mathbf{H}\mathbf{W}\mathbf{D}_e^{-1}\mathbf{H}^T\mathbf{X}^{(l)}\mathbf{\Theta}), \tag{5}$$

where $\mathbf{W}$ and $\mathbf{\Theta}$ are weight matrices that need to be optimized.

Comparing Equations 4 and 5, it can be seen that CSP is a simplified special case of Hyper-Conv with the matrices $\mathbf{W}$ and $\mathbf{\Theta}$ realized as non-learnable identity matrics. As the proposed method runs only the "forward pass" of Hyper-Conv, we do not use the non-linearity σ in the basic variant of CSP.

4.5 Comparison with Label Propagation

The label propagation algorithm as intruduced in [35] is expressed for an ordinary graph (with edges connecting exactly 2 nodes) as

$$\mathbf{Y}^{(l+1)} = \alpha\mathbf{D}_v^{-1}\mathbf{A}\mathbf{Y}^{(l)} + (1-\alpha)\,\mathbf{Y}^{(l)}, \tag{6}$$

where $\mathbf{D}_v$ denotes the diagonal matrix of degrees of a graph and $\mathbf{A}$ stands for its adjacency matrix.

To compare Label propagation with CSP, let us first express the value of $\mathbf{H}\mathbf{D}_e^{-1}\mathbf{H}^T$ as

$$\left(\mathbf{H}\mathbf{D}_e^{-1}\mathbf{H}^T\right)_{i,j} = \sum_k \frac{1}{\delta\left(e_k\right)}\mathbf{H}_{i,k}\mathbf{H}_{j,k}, \tag{7}$$

which represents for each pair of nodes the number of hyperedges connecting them, normalized by their degrees. Specifically, for an ordinary graph, this becomes

$$\mathbf{H}\mathbf{D}_e^{-1}\mathbf{H}^T = \frac{1}{2}\left(\mathbf{A} + \mathbf{D}_v\right). \tag{8}$$

With this simplification for ordinary graphs, equation 4 becomes

$$\mathbf{X}^{(l+1)} = \frac{1}{2}\mathbf{D}_v^{-1}\mathbf{A}\mathbf{X}^{(l)} + \frac{1}{2}\mathbf{X}^{(l)}. \tag{9}$$

which is equivalent to equation 6 with $\mathbf{X} = \mathbf{Y}$ (or a masked version thereof) and $\alpha = \frac{1}{2}$. CSP is in this instance therefore a generalization of label propagation with this particular value of α to hypergraphs (for generalization with arbitrary values of α, see Section 4.7). There is, however, another compelling reason to use CSP over Label propagation as presented in equation 6. The matrix multiplication $\mathbf{H}\mathbf{D}_e^{-1}\mathbf{H}^T$ does not preserve the sparsity of $\mathbf{H}$, which is typical for large datasets. Therefore the proposed implementation can be significantly more efficient than Equation 6, despite them being mathematically equivalent.

4.6 Comparison with the naive Bayes classifier

The naive Bayes classifier is a well know classification method which calculates posterior probability on y with a feature vector ξ as $p(y|\xi)$ using Bayes rule as $p(y|\xi) = \frac{p(\xi|y)p(y)}{p(\xi)}$ with the "naive" assumption that $p(\xi|y) = \prod_i p(\xi_i|y)$, where $p(\xi_i|y)$ is estimated on training set for all pairs of features and labels.

In the case of a multinomial Naive Bayes, the term $p(\xi_i|\mathbf{Y})$ is given by the empirical distribution observed on the training set. In the CSP setup, we therefore consider the hyperedges of $\mathcal{H}$ to represent the features ξ, that is $\mathbf{X} = \mathbf{H}$. In the binary case, the empirical distribution $p(\xi_i|\mathbf{Y})$ is equal to averaging the node signal across each hyperedge as described in Equation 1. The aggregation (product) then nicely demonstrates the difference between the filtering (convolutional) approach of CSP and the Bayesian (Maximum a posteriori) approach.

4.7 Convolutional Signal Propagation Variants

The comparison with the methods presented in Sections 4.4, 4.5 and 4.6 naturally suggests several alternative variants and generalizations of the basic CSP scheme.

Alternative normalizations of the adjacency matrix In graph neural networks, the adjacency matrix $\mathbf{A}$ is normalized by multiplying it with the inverse of the node degree matrix $\mathbf{D}_v$. While the DeepWalk algorithm [29] corresponds to the row-wise normalization $\mathbf{D}_v^{-1}\mathbf{A}$, newer methods also consider the column-wise normalization $\mathbf{A}\mathbf{D}_v^{-1}$ and most predominantly the symmetric normalization $\mathbf{D}_v^{-1/2}\mathbf{A}\mathbf{D}_v^{-1/2}$ introduced in [17]. Because the matrix $\mathbf{H}\mathbf{D}_e^{-1}\mathbf{H}^T$ plays in CSP a role similar to

the adjacency matrix $\mathbf{A}$ in GCN (see Equation 7), we can also consider the alternative column-wise normalized version of CSP:

$$\mathbf{X}^{(l+1)} = \mathbf{H}\mathbf{D}_e^{-1}\mathbf{H}^T\mathbf{D}_v^{-1}\mathbf{X}^{(l)} \tag{10}$$

and the symmetrically normalized version

$$\mathbf{X}^{(l+1)} = \mathbf{D}_v^{-1/2}\mathbf{H}\mathbf{D}_e^{-1}\mathbf{H}^T\mathbf{D}_v^{-1/2}\mathbf{X}^{(l)}. \tag{11}$$

Generalization of label propagation with general values of α While Section 4.5 shows that CSP is generalization of label propagation with $\alpha = \frac{1}{2}$ to hypergraphs, a more general variant of CSP allowing for $\alpha \in (0, 1)$ can be defined as

$$\mathbf{X}^{(l+1)} = 2\alpha\mathbf{D}_v^{-1}\mathbf{H}\mathbf{D}_e^{-1}\mathbf{H}^T\mathbf{X}^{(l)} + (1 - 2\alpha)\,\mathbf{X}^{(l)}. \tag{12}$$

This version is a full generalization of label propagation as described in Equation 6 to hypergraphs (equivalence can be proved by applying Equation 7).

All of the above variants of CSP can be implemented in a straightforward way by modifying Algorithms 1 and 2, without requiring full matrix multiplication.

5 Experimental evaluation

The goal of our experiments is to compare performance and execution time of the proposed Convolutional Signal Propagation (CSP) method with several well established baseline methods as well as with a simple Hypergraph Neural Network (HGCN) on a variety of real-world datasets from multiple domains. In this section, we introduce the considered datasets, reference methods, and tasks used for evaluation. Our aim is to validate the comparable performance of the proposed method while highlighting its low execution time. While we discuss the various variants of the proposed method in Section 4.7, a comprehensive evaluation of these variants will be conducted in a future work.

5.1 Datasets

We evaluated our methods on datasets from three distinct areas:

- **Citation datasets**: This category includes the PubMed, Cora, and DBLP datasets in their hypergraph variants, as introduced in [5]. In these datasets, the hyperedges may be defined in two distinct ways - either by a common author, or by co-citation, yielding two variants of the dataset. Each publication is labeled based on its topic.
- **Natural Language Processing (NLP)**: We used the Coronavirus tweets NLP dataset [25], which contains Twitter posts about COVID-19. The tweets were tokenized, with each tweet representing a node, labeled by its sentiment. Hyperedges are formed by tokens appearing in multiple tweets. The tokens are trained on the whole corpus with given number of tokens[3] (1000) using Sentencepiece algorithm [19]. Note that we can control the graph size with (number of hyperedges) by the parameter of tokenization.

[3] The number differs to the number of hyperedges from Table 1, due to special reserved tokens that are not used in corpus.

Table 1. Overview of datasets with their basic characteristics. $\Sigma_E = \sum_{e \in E} \delta(e)$ or equivalently the number of non-zero elements in the incidence matrix $\mathbf{H}$. Isolated nodes are nodes that are not connected by any hyperedge. Percentage of isolated nodes is their fraction related to the overall number of nodes.

Dataset	**CiteSeer**	**Cora-CA**	**Cora-CC**	**DBLP**	**PubMed**	**Corona**	**movie-RA**	**movie-TA**
Nodes	3312	2708	2708	41302	19717	44955	62423	62423
Isolated nodes	1854	320	1274	0	15877	0	3376	17172
Isolated node ratio	56%	12%	47%	0%	81%	0%	5%	28%
Hyperedges	1079	1072	1579	22363	7963	998	162541	14592
$\mathbf{\Sigma}_E$	3453	4585	4786	99561	34629	3455918	25000095	1093360
Average $\mathrm{d}(v)$	2.37	1.92	3.34	2.41	9.02	76.88	423.39	24.16
Average $\delta(e)$	3.20	4.28	3.03	4.45	4.35	3463	153.8	74.9
Classes	6	7	7	6	3	5	20	20

– **Recommender Systems**: We used the MovieLens 25M dataset [12], which contains 25 million movie ratings and one million tag applications. The nodes are individual movies labeled by their genres (allowing for multiple labels for one node). The hyperedges correspond to users and may again be defined in two ways, giving us two versions of the dataset: Either by a user rating multiple movies, or by a user assigning tags to multiple movies.

An overview of datasets including their basic characteristics is shown in Table 1. Due to the interpolative nature of CSP, each multiclass dataset is transformed into a series of binary datasets, where each class is treated as the positive class, and all other classes are treated as negative. The results are then averaged over all such obtained binary datasets.

We consider only structural information that is available in the hypergraph. No other information (such as node or hyperedge features) is included.

5.2 Tasks

We address two primary tasks in our experiments: transductive node classification and retrieval.

Classification Task

– **Aim**: Prediction of binary labels on the testing set
– **Method**: We use leave-one-out cross-validation with 10 folds, where nodes are randomly assigned to folds. One fold is hidden for testing, and the method is trained on the remaining nine folds. For each dataset, we generate test predictions for each node (when it is in the testing set). For each class, we calculate the ROC-AUC and average these scores. This average is reported as the classification score for each method on the given dataset.

Retrieval Task

– **Aim**: Ranking of nodes in the testing set maximizing the precision in the top positions
– **Method**: Folds are assigned in the same way as in the classification task, with one fold being used for training and the other 9 used as the testing dataset. The training set consists of all nodes in the graph, with the positive nodes in the training fold labeled as positive and all other

nodes labeled as unknown. The model is trained on this training set. For models that require negative training examples, we randomly sample a set of the same size as the testing set and consider these labels as negative. Although this introduces some label noise, we assume that the negative class is dominant, making the noise acceptable. The model then ranks the nodes from the testing folds, and we evaluate precision at the top 100 positions (P@100). This evaluation is performed for each fold and class, and the average P@100 over folds and classes is reported as the retrieval score for each method on the given dataset.

5.3 Evaluated Methods

We compare the following methods in our experiments. The variants of the CSP mentioned in Section 4.7 are not reported in this work. In future, we would like also to compare multiple choices of feature representation for reference methods on top of NMF. Nevertheless as our goal is just to show a competitive performance of CSP, we believe that it is sufficient to compare it with baseline methods without their fine-tuning.

- **The proposed CSP method**: Evaluated with 1, 2, and 3 layers, where we consider binary training labels as $\mathbf{X}^0$. After application of a given number of CSP layers (see Equation 4), the yielded (score) vector $\mathbf{X}^l, l \in \{1, 2, 3\}$ is used for both retrieval (top-100 scored test nodes) and for binary classification (with a given threshold on score).
- **Multinomial Naive Bayes**: Operates on one-hot feature vectors derived from hyperedges.
- **Random Forest, Logistic Regression, and HGCN**: These methods operate on feature vectors obtained from non-negative matrix factorization (NMF) of the incidence matrix[20][4] $\mathbf{H}$, with 10 iterations and a dimension of 60.
- **Method Settings**:
  - Random Forest, Logistic Regression, and Naive Bayes are used with their default settings.
  - A single layer HGCN implements Equation (5) with an output layer of dimension 2 and sigmoid non-linearity. We use logistic loss and train all datasets for 15,000 epochs using the Adam optimizer with default settings.
- **Random Baseline**: Included for comparison.

The results of these methods are evaluated and compared based on their performance on the classification and retrieval tasks across the datasets.

5.4 Classification Result

Table 2 lists the ROC-AUC for all methods on all datasets. Due to the numerical intensity of HGCN, only 5 folds were evaluated, and for the Movies dataset, only 4 out of 20 classes were considered. Prediction on isolated nodes was nearly random as only structural information was used, likely contributing to the relatively weak performance of all methods on the PubMed dataset. Since CSP handles only binary labels, reference methods were translated to a one-vs-other scenario, even though they can handle multi-class classification directly. Feature extraction using Non-negative Matrix Factorization (NMF) was not fine-tuned for each dataset, potentially impacting the performance of NMF-based baselines. Naive Bayes emerged as the strongest baseline, as it does not

[4] As an alternative to the NMF, we evaluated also representation generated by Laplacian positional encoding [6]. As the results were worse compared to NMF, we decided to not include them in the results.

require any feature preprocessing and works directly with the one-hot encoded incidence matrix. CSP was evaluated in three variants based on the number of layers, with the best choice varying by dataset. On the largest datasets (Corona and Movies), the best variant of CSP achieved performance comparable to the strongest competing baseline. Overall, CSP demonstrated comparable results with reference baselines. In larger datasets, where parameter tuning of baselines is more challenging, CSP proved to be one of the best-performing methods. Overall, CSP with fewer layers fared comparatively better than a version with multiple layers. We attribute this at first glance counter-intuitive result to the fact that the training set is fairly dense in the graph, which ensures sufficient information for all nodes even with fewer layers, while at the same time multiple layers may contribute to oversmoothing of the signal. CSP's simplicity and parameter-free nature confirm its suitability as a first-choice baseline method for classification tasks.

Table 2. The ROC-AUC for the classification task, averaged over all classes and all folds. The best method for each dataset is denoted by bold text, with methods within 0.05 underlined.

Method	CiteSeer	Cora-CA	Cora-CC	DBLP	PubMed	Corona	movie-RA	movie-TA
CSP 1-layer	0.646	0.882	0.716	0.968	0.537	**0.704**	0.789	0.717
CSP 2-layer	0.630	0.872	0.686	0.972	0.518	0.618	0.700	0.697
CSP 3-layer	0.613	0.862	0.655	0.972	0.516	0.580	0.640	0.673
Naive Bayes	**0.686**	**0.913**	0.775	**0.974**	**0.633**	**0.704**	0.753	0.557
HGCN-NMF	0.659	0.832	**0.786**	0.775	0.624	0.622	0.794	**0.724**
LR-NMF	0.604	0.794	0.703	0.705	0.556	0.647	0.754	0.675
RF-NMF	0.667	0.897	0.772	0.905	0.623	0.617	**0.797**	0.691
Random	0.499	0.505	0.489	0.501	0.502	0.503	0.499	0.487

5.5 Retrieval Results

Table 3 lists the P@100 for all methods on all datasets. The evaluation of HGCN and the Movies dataset in the retrieval task is restricted similarly to the classification task. Isolated nodes no longer cause a performance drop as long as there are sufficient number of non-isolated nodes in each class. Naive Bayes' performance is not as superior in this scenario as in classification task since the training set contains only positive nodes, preventing it from leveraging prior distribution knowledge about the target class. CSP, which does not use prior knowledge about the target class distribution, works very well on small datasets with lower average degree of the nodes and edges. In case of datasets with higher average degree of nodes (Movies), CSP does not extract the structural information as well as NMF and therefore the methods utilizing the features from NMF (mainly logistic regression) work much better excepting HGCN, which suffers from over-smoothing. In summary, CSP achieves superior performance for 4 of 8 datasets on retrieval task.

5.6 Complexity Evaluation

To evaluate the complexity of the studied methods, two approaches were taken – theoretical analysis of asymptotic complexity, and direct measurement of wall clock time to compute the methods on the available datasets. The asymptotic computational complexities of the used methods are

Table 3. The P@100 for the retrieval task, averaged over all classes and all folds. The best method for each dataset is denoted by bold text, with methods within 0.05 underlined.

Method	CiteSeer	Cora-CA	Cora-CC	DBLP	PubMed	Corona	movie-RA	movie-TA
CSP 1-layer	0.494	0.703	0.530	0.869	0.798	**0.530**	0.334	0.156
CSP 2-layer	0.558	0.718	0.681	0.865	0.826	0.440	0.336	0.186
CSP 3-layer	**0.568**	**0.721**	**0.707**	0.869	0.850	0.332	0.238	0.186
Naive Bayes	0.471	0.686	0.491	**0.951**	0.860	0.446	0.216	0.153
HGCN-NMF	0.482	0.671	0.607	0.794	**0.871**	0.392	0.257	0.148
LR-NMF	0.329	0.603	0.372	0.602	0.735	0.397	**0.580**	**0.356**
RF-NMF	0.303	0.474	0.482	0.843	0.794	0.381	0.470	0.131
Random	0.153	0.132	0.129	0.155	0.308	0.180	0.040	0.055

Table 4. Asymptotic complexity of the compared algorithms. For algorithms with multiple layers, only one layer is considered. n is the number of nodes, m is the number of hyperedges d is the signal dimensionality, T is the number of epochs, h is the number of hidden units, Σ_V is the sum of node degrees and Σ_E is the sum of hyperedge degrees.

CSP	Naive Bayes	HGCN	LR	RF
$\mathcal{O}\left(d\left(\Sigma_V + \Sigma_E\right)\right)$	$\mathcal{O}\left(nm\left\|V_{\text{train}}\right\|\right)$ [9]	$\mathcal{O}\left(T\Sigma_E hd\right)$ [33]	$\mathcal{O}\left(d\left\|V_{\text{train}}\right\|\right)$ [27]	$\mathcal{O}\left(d\log d\left\|V_{\text{train}}\right\|\right)$ [3]

Table 5. The execution time of a single retrieval task in miliseconds, averaged over all classes and all folds. The non-negative matrix factorization was excluded from the execution time.

Method	CiteSeer	Cora-CA	Cora-CC	DBLP	PubMed	Corona	movie-RA	movie-TA
CSP 1-layer	**1.35**	**1.23**	**1.24**	**3.51**	**4.01**	**22.83**	170.63	**12.07**
CSP 2-layer	2.41	2.25	2.24	7.46	7.35	47.39	349.67	25.88
CSP 3-layer	3.31	3.09	3.08	9.65	9.1	70.98	506.1	35.75
Naive Bayes	2.59	2.73	2.54	11.16	5.35	89.3	1 051.53	35.61
HGCN-NMF	23 714	23 690	23 709	29 123	23 879	112 314	620 717	70 778
LR-NMF	44.45	51.59	49.54	64.96	44.69	56	**61.82**	74.07
RF-NMF	140.76	143.85	134.52	1 148.83	323.6	1 608.58	697.85	716.68

listed in Table 4. Note that the non-negative matrix factorization is excluded from the table and has on its own a complexity of $\mathcal{O}\left(n^4 m^4\right)$, dominating the methods themselves. The execution time for individual methods is presented in Table 5. We evaluated the methods using standard implementations that would be widely used by practitioners. In particular, we used the Scikit-Learn [28] implementation with default settings for logistic regression, Naive Bayes and random forest. A Polars variant of Algorithm 2 was applied for the proposed CSP method. All these methods were executed on a GPU (Amazon EC2 G4 instance). The HGCN was executed using PyTorch-geometric [8].

Comparing the execution times in Table 5, HGCN is the most numerically complex method. Although methods to improve training efficiency, such as batching, are available, they were not considered in this work. Logistic regression and random forest exhibit comparable or shorter execution times in Corona and Movies datasets compared to CSP and Naive Bayes, largely because the most challenging part—extraction of structural information—is handled by nonnegative matrix factorization, which is not included in the reported times. The proposed CSP excels particularly in graphs with a low average degree of nodes, similar to Naive Bayes. The measured execution time of CSP aligns with its expected asymptotic complexity 4 and appears to be linear with the number of CSP layers, as anticipated.

In summary, HGCN is by far the most numerically intensive method and shows potential in some examples; however, there is still a significant amount of fine-tuning needed to achieve superior performance across datasets. NMF-based methods work exceptionally well on the Movies dataset for the retrieval task, though further tuning is required to properly extract structural information. Compared to Naive Bayes, CSP is even slightly simpler and parameter-free. In some problems, CSP outperformed Naive Bayes, and vice versa. Thus, it is definitely a good idea to consider both CSP and Naive Bayes when establishing baselines for tasks on structural data.

6 Conclusion

This paper presents a signal propagation algorithm for hypergraphs, termed Convolutional Signal Propagation (CSP). We formally describe the CSP algorithm and demonstrate its simplicity and efficiency of implementation. This formal description allows us to show clear relationships between CSP and well-known algorithms such as Naive Bayes, label propagation, and hypergraph convolutional networks. These relationships suggest various algorithmic variants, which we left for detailed future exploration.

We discuss the application of CSP to different types of signals. Our primary focus is propagating binary labels, which is used for classification and retrieval tasks, positioning CSP as a hypergraph variant of traditional label propagation. Additionally, propagating node features instead of labels as signals leads to feature propagation. This dual functionality showcases the versatility of CSP in handling various tasks on hypergraphs.

Our evaluation of CSP on several real-world datasets from multiple domains demonstrates its competitive performance in node classification and retrieval tasks compared to a range of reference methods. Furthermore, we compare the numerical complexity of these methods by examining their execution times, highlighting the simplicity and efficiency of CSP. This combination of competitive performance and low computational complexity makes CSP a promising approach for applications involving hypergraph-based data.

Acknowledgments. This work was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS20/183/OHK4/3T/14. Large language models were employed for proofreading,

drafting, and clarifying the text; however, the authors always validated and take responsibility for the final result.

Disclosure of Interests. All authors are currently employed by Cisco Systems, Inc. and conducted research presented in this work as part of their job. Pavel Procházka and Lukáš Bajer also own stock in Cisco Systems, Inc. The authors declare that their employment and stock ownership does not influence the objectivity of the presented research.

References

1. Bai, S., Zhang, F., Torr, P.H.S.: Hypergraph convolution and hypergraph attention. Pattern Recognition **110**, 107637 (Feb 2021). https://doi.org/10.1016/j.patcog.2020.107637, https://www.sciencedirect.com/science/article/pii/S0031320320304404
2. Beaini, D., Passaro, S., Létourneau, V., Hamilton, W., Corso, G., Lió, P.: Directional Graph Networks. In: Proceedings of the 38th International Conference on Machine Learning. pp. 748–758. PMLR (Jul 2021), https://proceedings.mlr.press/v139/beaini21a.html, iSSN: 2640-3498
3. Biau, G.: Analysis of a random forests model. The Journal of Machine Learning Research **13**(1), 1063–1095 (2012), https://www.jmlr.org/papers/volume13/biau12a/biau12a.pdf, publisher: JMLR. org
4. Carter, K.M., Idika, N., Streilein, W.W.: Probabilistic threat propagation for network security. IEEE transactions on information forensics and security **9**(9), 1394–1405 (2014)
5. Chien, E., Pan, C., Peng, J., Milenkovic, O.: You are AllSet: A Multiset Function Framework for Hypergraph Neural Networks. In: 10th International Conference on Learning Representations (ICLR 2022) (Oct 2021), https://openreview.net/forum?id=hpBTIv2uy_E
6. Dwivedi, V.P., Joshi, C.K., Luu, A.T., Laurent, T., Bengio, Y., Bresson, X.: Benchmarking Graph Neural Networks. Journal of Machine Learning Research **24**(43), 1–48 (2023), http://jmlr.org/papers/v24/22-0567.html
7. Feng, Y., You, H., Zhang, Z., Ji, R., Gao, Y.: Hypergraph Neural Networks. Proceedings of the AAAI Conference on Artificial Intelligence **33**(01), 3558–3565 (Jul 2019). https://doi.org/10.1609/aaai.v33i01.33013558, https://ojs.aaai.org/index.php/AAAI/article/view/4235, number: 01
8. Fey, M., Lenssen, J.E.: Fast Graph Representation Learning with PyTorch Geometric (Apr 2019), arXiv: 1903.02428
9. Fleizach, C., Fukushima, S.: A naive bayes classifier on 1998 kdd cup. Dept. Comput. Sci. Eng., University of California, Los Angeles, CA, USA, Tech. Rep (1998), http://cseweb.ucsd.edu/~cfleizac/cse250b/project1.pdf
10. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 855–864 (2016)
11. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. Advances in neural information processing systems **30** (2017)
12. Harper, F.M., Konstan, J.A.: The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems **5**(4), 19:1–19:19 (Dec 2015). https://doi.org/10.1145/2827872, https://doi.org/10.1145/2827872
13. Henne, V.: Label propagation for hypergraph partitioning. PhD Thesis, Karlsruher Institut für Technologie (KIT) (2015)
14. Huang, Q., He, H., Singh, A., Lim, S.N., Benson, A.R.: Combining Label Propagation and Simple Models Out-performs Graph Neural Networks (Nov 2020), arXiv:2010.13993
15. Jiang, J., Wei, Y., Feng, Y., Cao, J., Gao, Y.: Dynamic Hypergraph Neural Networks. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19. pp. 2635–2641. International Joint Conferences on Artificial Intelligence Organization (Jul 2019). https://doi.org/10.24963/ijcai.2019/366, https://doi.org/10.24963/ijcai.2019/366

16. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
17. Kipf, T.N., Welling, M.: Semi-Supervised Classification with Graph Convolutional Networks. In: 5th International Conference on Learning Representations, {ICLR} 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net, Toulon, France (Apr 2017), https://openreview.net/forum?id=SJU4ayYgl
18. Kschischang, F.R., Frey, B.J., Loeliger, H.A.: Factor graphs and the sum-product algorithm. IEEE Transactions on information theory **47**(2), 498–519 (2001)
19. Kudo, T., Richardson, J.: SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. In: Blanco, E., Lu, W. (eds.) Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. pp. 66–71. Association for Computational Linguistics, Brussels, Belgium (Nov 2018). https://doi.org/10.18653/v1/D18-2012, https://aclanthology.org/D18-2012
20. Lee, D., Seung, H.S.: Algorithms for non-negative matrix factorization. Advances in neural information processing systems **13** (2000)
21. Lee, G., Lee, S.Y., Shin, K.: VilLain: Self-Supervised Learning on Hypergraphs without Features via Virtual Label Propagation (May 2024), https://openreview.net/forum?id=ukHSfLQAB5
22. Levi, F.W.: Finite geometrical systems (1942)
23. Li, P., Wang, Y., Wang, H., Leskovec, J.: Distance encoding: Design provably more powerful neural networks for graph representation learning. Advances in Neural Information Processing Systems **33**, 4465–4478 (2020)
24. McKinney, W.: Data Structures for Statistical Computing in Python. In: Walt, S.v.d., Millman, J. (eds.) Proceedings of the 9th Python in Science Conference. pp. 56 – 61 (2010). https://doi.org/10.25080/Majora-92bf1922-00a
25. Miglani, A.: Coronavirus tweets NLP - Text Classification (2020), https://www.kaggle.com/datasets/datatattle/covid-19-nlp-text-classification
26. Ng, A., Jordan, M.: On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. Advances in neural information processing systems **14** (2001)
27. Nocedal, J., Wright, S.J. (eds.): Numerical Optimization. Springer Series in Operations Research and Financial Engineering, Springer-Verlag, New York (1999). https://doi.org/10.1007/b98874, http://link.springer.com/10.1007/b98874
28. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research **12**(85), 2825–2830 (2011), http://jmlr.org/papers/v12/pedregosa11a.html
29. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 701–710 (2014)
30. Prochazka, P., Dvorak, S., Bajer, L., Kopp, M., Shcherbin, K.: Cross-domain indicator of compromise (ioc) identification (Sep 2023), https://patents.google.com/patent/US20230281300A1/en
31. Rossi, E., Kenlay, H., Gorinova, M.I., Chamberlain, B.P., Dong, X., Bronstein, M.M.: On the Unreasonable Effectiveness of Feature Propagation in Learning on Graphs With Missing Node Features. In: Proceedings of the First Learning on Graphs Conference. pp. 11:1–11:16. PMLR (Dec 2022), https://proceedings.mlr.press/v198/rossi22a.html, iSSN: 2640-3498
32. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. arXiv preprint arXiv:1710.10903 (2017)
33. Yadati, N., Nimishakavi, M., Yadav, P., Nitin, V., Louis, A., Talukdar, P.: HyperGCN: A New Method For Training Graph Convolutional Networks on Hypergraphs. In: Advances in Neural Information Processing Systems. vol. 32. Curran Associates, Inc. (2019), https://proceedings.neurips.cc/paper/2019/hash/1efa39bcaec6f3900149160693694536-Abstract.html

34. Zhou, D., Huang, J., Schölkopf, B.: Learning with hypergraphs: Clustering, classification, and embedding. Advances in neural information processing systems **19** (2006), https://proceedings.neurips.cc/paper/2006/hash/dff8e9c2ac33381546d96deea9922999-Abstract.html
35. Zhu, X., Ghahramani, Z.: Learning from Labeled and Unlabeled Data with Label Propagation (Jul 2003)