Unsupervised Learning in Networks, Sequences and Beyond

by

Yike Liu

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Physics)
in The University of Michigan
2019

Doctoral Committee:

Associate Professor Jieping Ye, Chair
Professor Charles Doering
Assistant Professor Danai Koutra
Professor Mark Newman
Assistant Professor Kevin Wood

Yike Liu
yikeliu@umich.edu
ORCID iD: 0000-0003-0117-1656

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor Prof. Jieping Ye, who has shared unlimited resources and opportunities with me. It will be impossible for me to have the chance of learning all the exciting topics, going to inspiring talks and being acquainted with amazing friends without him. He has provided a platform for me to explore, learn and practice that I would have not imagined. His perseverance towards work, curiosity on research, and kindness to people are legacies I will cherish for the rest of my life.

I would like to express my gratitude to Prof. Danai Koutra, who have taught me how research is done from the very beginning. I have learned precious lesson from working with her. Together I give thanks to all my committee members, who helped me through the entire dissertation.

Many thanks to my mentors during all my internships: Dr. Linhong Zhu, who has helped me ever since my first internship, is always there to encourage me and never stingy on giving compliments about me; Dr. Li Deng and Dr. Pusheng Zhang, whom I have spent a perfect summer in Seattle with, and learned a great deal from, you

are the ones who opened the door of NLP for me; and Prof. Kevin Knight, who have showed faith in me and boosted my knowledge in the NLP domain. There is more gratitude than words can express and I will thank them my taking what I learned from them into the work I will do in the future.

I would like to thank each and one of my friends, who have been a treasure at every moment of my life. My dear friends Zhiheng Zuo and Qiaozhi Song, are the sweetest girls who always have faith in me and there to help. Dr. Kunlei Zhang, Chang Zhou, and Jianhua Zhang, who are not only friends to keep me company but also turn to whenever and for whatever I need. Dr. Yuqing Kong, who plays the role of not just the expert of mathematics but also a warm guide in the most frustrating time of my life. And many others who have been on my way towards today.

Special thanks to my partner, Haojun Ma, who has provided the most solid support ever in every single moment of our time together.

Last but not least, great thanks to my parents, Ling Tong and Bogao Liu, even across oceans I am constantly reminded they have the warmest spot for me in the world.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . . . **ii**

LIST OF TABLES . . . . . **vi**

LIST OF FIGURES . . . . . **vii**

ABSTRACT . . . . . **ix**

CHAPTER

I. Introduction 1

II. Unsupervised Learning In A Nutshell 4

2.1 Data Types 4
2.2 Task Types 5
2.3 Model Types 9
2.4 Goals and Applications 10

III. Networks: Summarization And Clustering 13

3.1 CondeNSe: Graph Summarization 13
3.1.1 CondeNSe: Proposed Model 20
3.1.2 CondeNSe: Our Proposed Algorithm 29
3.1.3 Empirical Analysis 37
3.2 Network Clustering 47

IV. Sequences: Networks And Neural Models 53

4.1 CCTN: Coupled Clustering of Time-Series and Networks 53
4.1.1 Proposed Problem 57
4.1.2 CCTN-inc: Incremental Updates 63
4.1.3 Experiments 66
4.1.4 Case Study on Real Data 74

4.2 Recurrent Deep Learning Games: Optimizing RNN Training 76
4.2.1 RDLP : Recurrent Deep Learning Problem 79
4.2.2 Analysis And Improvements 88
4.3 Adversarial Learning 89

V. Power of Unstructured Data 91

5.1 Learning Generic Embeddings For Entities 93
5.1.1 Problem Formulation 94
5.1.2 Experiment Setup 94
5.1.3 Methods 96
5.1.4 Evaluation 97

VI. Conclusion 102

APPENDIX **104**

BIBLIOGRAPHY **117**

LIST OF TABLES

Table

2.1 Qualitative comparison of related approaches. 6

3.1 Qualitative comparison of the graph clustering techniques included in CondeNSe. Symbols: n = number of nodes, m = number of edges, k = number of clusters/partitions, t = number of iterations, d = average degree, $h(m_h)$ = number of nodes (edges) in hyperbolic structure. 18

3.2 Major symbols and definitions. 21

3.3 Summary of graphs used in our experiments. 37

3.4 CondeNSe: Compression rate with respect to the empty model. In parentheses, number of structures in the corresponding summary. A "-" means that the corresponding method was terminated after 4 days. (* In the interest of time, the summary size was limited to 15.) 38

3.5 Overlapping supernode pairs and average similarity in parentheses. CondeNSe reduces the overlap. A "-" means that the corresponding method was terminated after 4 days. 39

3.6 CondeNSe: Number of structures per type in the summaries in the format $[fc, st, bc]$, for VoG we have $[fc + nc, st, bc + nb]$, where nc is near-clique and nb is near-bipartite core. The CondeNSe summaries are more balanced, *without* a specific pattern type dominating in all the graphs. In the interest of time, we find the top-50 and top-15 structures for `Enron` and `EUmail`, respectively. A "-" means that the corresponding method was terminated after 4 days. 41

3.7 Agreement of Step and its variants. They approximate Step quite well. (* Agreement based on the top-50 structures due to Step-P's lack of scalability.) 43

3.8 CondeNSe as an evaluation metric: Compression rate of clustering methods with respect to the empty model (i.e., percentage of bits for encoding the graph given the chosen model vs. the empty model). 44

3.9 Ablation study for `AS-Oregon`. Louvain and SlashBurn contribute most to the CondeNSe summaries. 46

4.1 Major symbols and definitions. 58

4.2 Synthetic data: Description of the six cases that we designed for evaluation. For each cluster, we generate for its constituent nodes a specific type of time series per case (e.g., identical, correlated, noisy). 66

4.3 Real data 68

LIST OF FIGURES

Figure

3.1 CondeNSe generates simpler and more compact supergraphs than baselines. Yellow, red, and green nodes for stars, cliques, and bipartite cores, respectively. 15
3.2 Illustration of MDL encoding. 23
3.3 `Choc`: CondeNSe-Step generates more compact supergraphs. (b)-(c): The full supergraphs by VoG-GnF, and CondeNSe-Step, resp. Yellow for stars, red for cliques, green for bipartite cores. The edge weights correspond to the number of inter-supernode edges. 39
3.4 Node coverage vs. edge coverage—marker size corresponds to the graph size. Step variants have better node coverage, and handle the summary coverage-conciseness trade-off well. 40
3.5 Runtime vs. # of edges: k-Step is more efficient than the other methods, and scales to larger graphs. 42
3.6 The agreement is robust to the number of partitions, while the runtime decreases. . 45
3.7 Number of structures found by the clustering methods for `AS-Oregon`. Transparent/solid rectangles for before/after the structure selection step. Notation: fc: full clique, st: star, ch: chain, bc: bipartite core, hs: hyperbolic structure. 45
4.1 Human trafficking example: Nine phone numbers (nodes) form three clusters with tight connections, and similar temporal behaviors. The matrix on the left illustrates the representation of the network structure ($\mathbf{A}$) and the ones on the right show the different types of node-specific behaviors over time ($\mathbf{X}_k$). 54
4.2 Accuracy of CCTN and CCTN-inc on synthetic and real data. 69
4.3 MITRE: CCTN-inc approximates CCTN well. They yield similar clusterings. . . 71
4.4 Runtime analysis. 72
4.5 Advertisements related to the phone numbers 1-3236****** and 1-3232******. . . 75
4.6 The computational graph to compute the training loss of a recurrent network that maps an input sequence of $\mathbf{x}$ values to a corresponding sequence of output $\mathbf{o}$ values. Figure from [GBC16] 78
4.7 Block diagram of the LSTM recurrent network cells.Figure from [GBC16] 85
4.8 Training Loss vs. Iterations 87
5.1 Workflow of experiment on user embedding extraction. 95
5.2 Word2vec: 2D embedding. Each profile feature is clearly distinguished by the embedding. 99

5.3 Language model: 2D embedding. Each profile feature is clearly distinguished by the embedding. 100
A.1 Rand index-based accuracy of CCTN and CCTN-INC on synthetic and real data. 110
A.2 MITRE: CCTN-INC approximates CCTN well based on both the NMI and rand index metrics. 111
A.3 CCTN is robust with respect to the parameters. The different lines in the plots correspond to different synthetic cases, as described in the rightmost plot (d). . . . 111

ABSTRACT

Unsupervised learning has gained tremendous interest in the past decade in various research communities, by virtue of its capability of exploiting unlabeled data and discovering patterns. The output of classical unsupervised models such as clustering and summarization can serve for the purposes of interpreting as well as preprocessing data for downstream tasks. Unsupervised models also present great potential in representing unstructured data and generalizing to unobserved data. In this thesis, I show the power of unsupervised learning in different data formats - networks and sequences, where models are formed in various ways. For networks, I introduce CondeNSe for summarizing large graphs in an unsupervised way, as well as its relations to graph clustering. I extend the clustering problem to sequences and propose to solve a coupled clustering problem on graphs and sequences. In the sequence modeling domain, I discuss my study on predicting sequences in the context of game theory, which falls in the domain of adversarial learning. Finally, I switch gears to structured data and demonstrate the power of unsupervised models on unstructured data in representing and extracting information.

CHAPTER I

Introduction

Unsupervised learning, normally in comparison with supervised learning, refers to a subset of machine learning tasks where training data is not labeled. Models must learn relationships between elements in a data set, possibly in forms of hidden structures, patterns or features to represent the data [Dee18].

Common tasks/algorithms in unsupervised learning include but are not limited to: clustering, anomaly detection, learning latent variable models, and neural-network-based methods. I will briefly discuss these algorithms and present my contributions to clustering and neural-network-based methods in detail.

Clustering, or finding groups of similar entities, is a fundamental task in data mining and machine learning, with applications in human mobility analysis, sensor data analytics in healthcare, intelligent urban systems, climate monitoring, and more. Depending on the notion of a cluster, data format and domain, clustering algorithms can take on different forms. For example, clustering methods have been studied on both time series, networks etc. [BUWK15, JRSD15, DDT$^+$16, Hes04, BGLL08]. There are also applications where different data forms co-occur and the clustering is performed on them jointly. Many clustering algorithms stand out as popular tools for preliminary data analysis or preprocessing steps, to name a few: hierarchical clus-

tering [Joh67], k-means, mixture models [MLR19], DBSCAN [EKS+96] etc. Many surveys have been conducted on clustering extensively [XW05, Ber06, Lia05] while posing many interesting future directions.

There are numerous unsupervised learning models research for anomaly detection and learning latent variables in the literature: anomaly detection has been a problem widely studied in the data mining community [BKNS00, ZF18, CBK09, ATK15], on both its definition and mining strategy; a lot of approaches have been proposed for learning latent variable models, e.g. Expectation-maximization algorithm, principle component analysis, and independent component analysis, which have alleviate the problem of high dimensions or feature correlation in many machine learning problems.

Neural Networks have emerged as the most popular type of models for machine learning tasks in the past a few years. By introducing simple and relational representations of data, deep learning models allow computers to learn from experience and understand the world in terms of a hierarchy of concepts, represented by layers of neurons with the connections between them [GBC16]. Specifically, many unsupervised deep learning models have been proposed for either learning the data representation, compressing data or accommodating specific tasks. E.g., autoencoders are designed to learn a representation/encoding for a set of data, where the input data is compressed by an encoder network and reconstructed by a decoder network [Bal87, RHW85]. Apart from representation learning, unsupervised deep learning serves many other tasks, e.g. Generative Adversarial Networks (GANs) [GPAM+14] takes advantage of two neural networks in a zero-sum game where they contest each other, for the purpose of generating real candidate data.

Without the data labels present, all the algorithms above converge to the idea of mimicking human logic on data observation and analysis. Specifically, taking the view of density estimation, supervised learning intends to infer a conditional probability distribution $P(x|y)$ where y is the label, while unsupervised learning infers a priori probability distribution $P(x)$. For this reason, unsupervised models usually present to be much more powerful in generalization and performing tasks across domains.

The thesis is organized as follows: Chapter II will introduce current state of the art of unsupervised learning with popular models as well as applications. Unsupervised learning in networks and sequences will be discussed in detail in Chapter III and IV respectively. In Chapter V I will show the power of unstructured data in unsupervised learning with specific models, and finally concluding in Chapter VI.

CHAPTER II

Unsupervised Learning In A Nutshell

Attention on unsupervised learning has boosted significantly over the past decade, with related research publications more increasing more than 10 times since 2008 [Dee18]. There is rich literature on surveys of related topics [PY10, GCB04, Zhu05, CS14], where we not only see unsupervised learning itself prosper, but also joining with other interesting topics such as transfer learning, semi-supervised learning, clustering, etc. Without stretching to cover every possible model, I'll provide a few dimensions to look at the state of the art unsupervised models, with focus on clustering and neural-network-based models, which are the main contributions of my thesis.

2.1 Data Types

First, it is natural to look at unsupervised learning from different data types. Other than being unlabeled, unsupervised learning has very limited constraint on the input data. Therefore, we see the same task solved with different type of data and very often many algorithms share similar ideas with adaptions. Take clustering for example. Most commonly we see input as a data set $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ consisting of N observations of D-dimensional variable $\mathbf{x}$, as we see in the setting of k-means clustering, where data is considered in a Euclidean space. Variations can be taken on the distance measure,

e.g. Chebyshev or Manhattan, for different applications. Graph mining community has looked deep into clustering on networks (sometimes referred to as community detection or graph partitioning) where the properties of networks come into play and are considered for the clustering algorithm [Sch07, For10, Dhi01]. Extending to the temporal space, clustering is heavily studied for time-series or sequential data [Lia05, ASW15, Cor88]. Recently research communities have began to focus on clustering of more complex data, normally formulated as a jointly clustering problem, e.g. spatio-temporal clustering [KMNR09], network-temporal clustering [LZS+19].

2.2 Task Types

Task type is one of the most important measures for distinguishing different unsupervised models. Major classical unsupervised tasks include: clustering, anomaly detection, and for some Bayesian models-learning latent variables. When it comes to neural models, they can perform many more and challenging tasks due to its end-to-end learning ability and great representation power.

Among different tasks, researchers have devoted efforts on them from many perspectives, e.g. problem formulation, methodology, evaluation, etc. Take clustering for example. For different type of data, researchers try to provide good measure of internal homogeneity and external separation for an optimal clustering [XW05]. Considerable amount of work has been put into evaluation of clusters as well as parameter selection (e.g. number of clusters) [AGAV09, Ran71, ZZX08]. Time-series clustering [Lia05] aims to group coherent time-series sequences together based on a certain similarity measure of either raw data or extracted features or representations [BUWK15, CP08]. Targeting imputation and prediction for co-evolving higher-order time series, [CTF+15] jointly models them in a 'network' of time series

and follows a tensor decomposition approach. This model is more restrictive than ours that loosely couples the network with the time dimension. I present a set of other representative works (not exhaustive) in Table 2.1. For an extensive review on graph clustering, we refer the interested reader to two surveys [Sch07, LSDK18]. Recent work [FZ16] converts the time-series clustering problem to graph clustering by first creating a similarity graph where each node corresponds to a time sequence and then applying modularity-based clustering [BGLL08]. Moreover, there has been increasing interest in jointly clustering attribute and relational data. For instance, Cho et. al. [CVSFG16] proposed a shared latent space model for describing both network and behavioral data. However, their work focused on static behaviors/attributes only. Also, [CBLH12] extracts subgraphs with similarly evolving *structural* patterns, but do not take into account non-structural temporal behaviors. Our work, on the other hand, explicitly addresses the temporal aspect of the problem by representing each node as a multivariate time series. In Chapter IV we will talk about my contributions to joint clustering in detail.

Table 2.1: Qualitative comparison of related approaches.

Methods	**Data Source**			**Similarity**	
	Raw	**Feature**	**Representation**	**Dist**	**Model**
[BUWK15, MLKCW03, KGP01, Dah96, PG15]	✓			✓	
[CP08, XY04]	✓				✓
[KP98, SK92, DDT$^+$16]			✓	✓	
[GHLR01]		✓		✓	
[OFC99]	✓			✓	✓

In the graph mining domain, clustering is constantly associated to the problem of graph summarization [LSDK18]. Most research efforts in graph summarization focus on plain graphs and can be broadly classified as group-based [LT10, RGM03], compression-based [CKL$^+$09, NRS08, GKSL17, KKVF14], simplification-based, influence-based, and pattern-based [CH94]. Dynamic graph summarization has been stud-

ied to a much smaller extent [SKZ+15, JK17b]. Beyond the classic definition of graph summarization, there are also approaches that summarize networks in terms of structural properties (e.g., degree, PageRank) by automatically leveraging domain knowledge [JK17a, JLS+17]. Most related to my work are the ideas of node grouping and graph compression. Built on these ideas, two representative methods, MDL-SUMMARIZATION [NRS08] and VOG [KKVF14], are MDL-based summarization methods that compress the graphs by findingnear-structures (e.g., (near-) cliques, (near-) bipartite cores). MDL-SUMMARIZATION, which iteratively combines neighbors into supernodes as long as it helps with minimizing the compression cost, includes mostly cliques and cores in the summaries, and has high runtime complexity.

On the other hand, VOG finds structures by employing SLASHBURN [KF11] (explained below) and hence is particularly biased towards selecting star structures. Moreover, it creates summaries (i.e., lists of structures) using a greedy heuristic on a pre-ordered set of structures. Unlike these methods, CONDENSE performs ensemble pattern discovery and handles edge-overlapping structures. Furthermore, its summary assembly is robust to the ordering of structures.

Anomaly detection is a major topic in both statistics community and data mining community. We refer interested readers to some comprehensive surveys [CBK09, ATK15, CB10] as it is not the major contribution of this thesis.

Models for learning learning latent variables is an important type of models of unsupervised learning. The idea is to learn a statistical model that relates a set of observables to a set of latent variables. Many of the clustering methods can be perceived as a latent variable model, and vice versa. E.g., k-means clustering is essentially an Expectation-maximization algorithm, spectral clustering is built on top of matrix

factorization that projects nodes to a latent space. Actually, the idea of separating signals - either by principle component analysis, or independent component analysis, or non-negative matrix factorization, and singular value decomposition - is deeply rooted in classical machine learning and in many algorithms data is projected to a latent space for analysis. Our algorithm in Chapter IV, CCTN (Coupled Clustering of Time-Series and Networks), also takes advantage of this idea.

Neural networks are powerful tools for unsupervised learning, for they are end-to-end models that capture information in complex data easily; and they provide natural representations of data for specific tasks (in supervised learning) and good generic representations in unsupervised learning. Many classical unsupervised algorithms can serve as a preprocessing step in many deep learning tasks. E.g., to alleviate the computational cost training convolutional networks, instead of directly a computing convolutional layer in both forward pass and backpropagation, one can apply k-means clustering to small image patches to obtain kernels [CN11]. However, deep learning models play significant roles in representation learning [GBC16]. A typical framework of training representations on unsupervised tasks before applying on supervised tasks has shown huge success in recent years, especially in natural language processing [PNI+18, DCLT18, RNSS18]. Numerous researchers have shown that unsupervisedly learned representations present robust performance across different modalities and domains, which is essential in transfer learning or even zero-shot learning. Goodfellow et. al. explained in detail how unsupervised pretraining (or greedy layer-wise unsupervised pretraining) works in detail [GBC16]. Relying on a single layer representation learning algorithm such as an RBM (Restricted Boltzmann machine), a single-layer autoencoder, a sparse coding model or any latent representation learning models, it trains greedily the output layers depending on the

previous layer, that are supposedly to be simpler than the previous one. NLP has been one of the fields that best benefit from unsupvised learning recently - a series of language models have been proposed that kept improving the performance of text representations: ELMo [PNI+18], GPT [RNSS18], and BERT [DCLT18]. Erhan et. al. [EBC+10] and Goodfellow et. al. [GBC16] provided insights on representation learning powers of unsupervised learning. In short, unsupervised pretraining roots on the idea unsupervised tasks essentially provides a way of learning about the input distribution without specific labels involved, as we mentioned in Introduction, hence leaving the representations with more information. And the pretraining results can be perceived as parameter initialization, which is equivalent to introducing a significantly effective regularization on the model.

2.3 Model Types

There is not consensus on a taxonomy of unsupervised model types within the machine learning community. I would like to take a traditional view on the models, which is far from comprehensive but not yet looked much into in this thesis - generative vs. discriminative models. Bishop gives a rather formal definition to distinguish them [Bis06]: approaches that explicitly or implicitly model the distribution of inputs as well as outputs are known as generative models, because by sampling from them it is possible to generate synthetic data points in the input space; approaches that model the posterior probabilities directly are called discriminative models. Clustering and anomaly detection algorithms normally falls into the discriminative category. Latent variable models are rather complicated, having considerable models on both sides. For example, most EM algorithms are generative as they maximize a posterior likelihood while trying to discover the hidden variable; KNN (k-nearest-neighbors)

is discriminative as it models a conditional probability. Distinction becomes more ambiguous for complex models like neural networks: language models are considered generative; generative adversarial networks [GPAM+14] are both generative and discriminative as there are two networks with corresponding objectives. It is important for us to keep aware of the part of models that is generative or discriminative, which helps us understand the intuition behind the model as well as how we can improve it.

2.4 Goals and Applications

Being unsupervised is far from having no objective. Unlike supervised learning, there is no target label for prediction, and the accuracy is difficult to evaluate. In order to model the data distribution, what contributes to a good objective? Graves and Ranzato provide some insights at the unsupervised deep learning tutorial at NeurIPS 2018 [AG18]. The key idea still lies in density modeling: maximum likelihood on data (without labels). Many generative models take this approach by assuming different prior distributions. The benefit of these generative models such as GANs [GPAM+14], is that we are able to simulate the generation of data, hence getting a view out of the sample space. Another important type are autoregressive models, which assumes a chain rule for probabilities, e.g. in language modeling, one assumes the probability as $P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t | w_{t-1}, \ldots, w_1)$, where $w_1, \ldots, w_T$ is the input data sequence. Targeting sequential data, it splits high dimensional data into small pieces and tries to predict each piece from those before. Autoregressive models have proven to work extremely well on sequential data as they are simple to define and it is easy to generate samples [VDODZ+16, OKK16, KvdOS+17]. Finally, taking the view of representation learning, a representation can be seen as

a description of the data, if any luck, we might be able to interpret it [SFH17]. One way to make these representations accessible is to compress them/force them through a bottleneck. Autoencoder is a model that implement this objective. Specifically, Minimum Description Length is exploited by VAE [Doe16], as well as many summarization method such as our CondeNSe [LSSK18] that we will discuss in detail in Chapter III. Mutual information is another metric often used to evaluate a data compression, contrastive predictive coding [OLV18] and noise-contrastive estimation [GH12] are both realizations of it. There are many models that extend these goals/objectives outside of unsupervised learning, either trying to combine with supervised learning, inspiring plenty of semi-supervised methods; or embedding in reinforcement learning tasks, which I am not elaborating here. Interested readers can always refer to the tutorial for more details [AG18].

Unsupervised learning is widely applied both for its own benefit and serving as a preprocessing step for supervised tasks. Clustering is commonly used as a tool for data partitioning: community detection in social networks [For10], market segmentation for targeting customers [PS83], and various applications in medical/biological data, e.g. brain fMRI data [HBC+92] or MR data [KBK00], gene clustering [LS03] with hierarchical clustering on different expression levels, and many other multidisciplinary applications such as climate indices derivation on earth science data, document clustering on content, etc. Anomaly detection methods are the core for fraud detection [KLSH04]. On the other hand, representation learning (both latent space and deep learning models) are the main tools for preprocessing noisy data, for multiple purposes such as dimension reduction, decorrelation, data interpretation, learning generation process, and processing data into structured form for downstream tasks. All of these are extremely important tasks in machine learning and data mining and

have been topics people study for decades. There is no doubt that unsupervised learning will keep prospering considering the quantity and heterogeneity of data we have as of today.

CHAPTER III

Networks: Summarization And Clustering

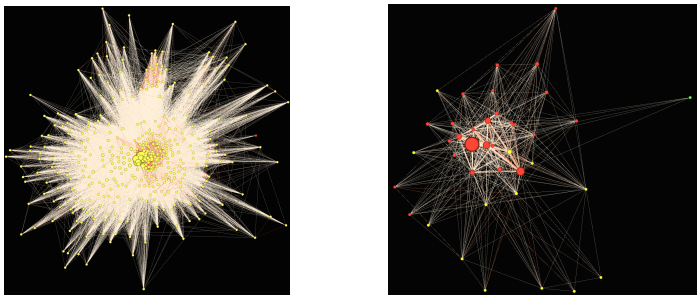
3.1 CondeNSe: Graph Summarization

Summarizing a large graph with a much smaller graph is critical for applications like speeding up intensive graph algorithms and interactive visualization. In this chapter, I will discuss my work CondeNSe (CONditional Diversified Network Summarization), a Minimum Description Length-based method that summarizes a given graph with approximate "supergraphs" conditioned on a set of diverse, predefined structural patterns. The majority of this section can be found in [LSSK18]. CondeNSe features a unified pattern discovery module and a set of effective summary-assembly methods, including a powerful parallel approach, k-Step, that creates high-quality summaries not biased toward specific graph structures. By leveraging CondeNSe's ability to efficiently handle overlapping structures, we contribute a novel evaluation of seven existing clustering techniques by going beyond classic cluster quality measures. Extensive empirical evaluation on real networks in terms of compression, runtime, and summary quality shows that CondeNSe finds 30-50% more compact summaries than baselines, with up to 75-90% fewer structures and equally good node coverage.

In an era of continuous generation of large amounts of data, summarization tech-

niques are becoming increasingly crucial to help abstract away noise, uncover patterns, and inform human decision processes. Here we focus on the summarization of *graphs*, which are powerful structures that capture a number of phenomena, from communication between people [LKF05, BHKL06, KKPF13] to links between webpages [kle99], to interactions between neurons in our brains [OCP14, YZD+19]. In general, graph summarization or coarsening approaches [LSDK18] seek to find a concise representation of the input graph that reveals patterns in the original data, while usually preserving specific network properties. Summarization is also studied as a way to speed up neural networks and increase robustness [YZD+19] as well as represent nodes in a latent space [JHRK19, JRK+18]. As graph summaries are application-dependent, they can be defined with respect to various aspects: they can preserve specific structural patterns, focus on some entities in the network, preserve the answers to a specific set of queries, or maintain the distributions of some graph properties. Graph summarization leads to the reduction of data volume, speedup of graph algorithms, improved storage and query time, and interactive visualization. Its major challenges are in effectively handling the volume and complexity of data, defining the interestingness of patterns, evaluating the proposed summarization techniques, and capturing network structural changes over time. The graph mining community has mainly studied summarization techniques for the structure of static, plain graphs [CKL+09, NRS08] and to a smaller extent, methods for attributed or dynamic networks [SKZ+15].

Our method, CondeNSe or CONditional Diversified Network Summarization, summarizes the structure of a given large-scale network by selecting a small set of its most informative structural patterns. Inspired by recent work [NRS08, KKVF14], we formulate graph summarization as an information-theoretic optimization problem



(a) Prior work [KKVF14] (b) Our method: CondeNSe-Step

Figure 3.1: CondeNSe generates simpler and more compact supergraphs than baselines. Yellow, red, and green nodes for stars, cliques, and bipartite cores, respectively.

in search of local structures that collectively minimize the description of the graph.

CondeNSe is a unified, edge-overlap-aware graph summarization method that summarizes a given graph with approximate "supergraphs" conditioned on diverse, predefined structural patterns. An example is shown in Figure 3.1, where the (super)nodes in Figure 3.1b correspond to *sets of nodes* in the original graph. Specifically, the predefined patterns include structures that have well-understood graph-theoretical properties and are found in many real-world graphs [kle99, AGMF14, FFF99, PSS+10]: cliques, stars, bipartite cores, chains, and patterns with skewed degree distribution.

Our work effectively addresses three main limitations of prior summarization work such as [KKVF14] and [KF17], namely: (i) its heavy dependence on the structural pattern discovery method and intrinsic tendency, or *bias*, to select star-like structures in the final summary; (ii) its inability to handle edge-overlapping patterns in the summary; and (iii) its dependence on the order in which candidate structures are considered for the final summary. Our proposed unified approach effectively handles these issues and results in robust, compact summaries with $5-10\times$ fewer structural patterns (or supernodes) and up to 50% better compression.

CondeNSe has three main modules that tackle the above-mentioned shortcomings: (i) A unified structural pattern discovery module leverages the strengths of various

popular graph clustering methods (e.g., LOUVAIN [BGLL08], METIS [KK99]) to address the structural biases that each method introduce in the graph summary; (ii) A Minimum Description Length-based (MDL) formulation with a penalty term effectively minimizes redundancy in edge coverage by the structural patterns included in the summary. This term is paramount when the candidate structural patterns have significant edge overlap, such as in the case of our unified structure discovery module; (iii) An iterative, multi-threaded, and divide-and-conquer-based summary assembly module reduces even more the bias during the summary creation process by being *independent* of the order in which the candidate structural patterns are considered. This parallel module is up to 53× faster than its serial version (on a 6-core machine).

Our contributions in this method are as follows:

• **Approach**: We introduce CONDENSE, an effective unified, edge-overlap-aware graph summarization approach with a powerful parallel summary assembly module (K-STEP) that creates compact and easy-to-understand graph summaries with high node coverage and low redundancy.

• **Novel Metric**: We propose a way to leverage CONDENSE as a proxy to compare graph clustering methods with respect to their summarization performance on large, real-world graphs, complementing the usual evaluation metrics in the related literature (e.g., modularity, conductance).

• **Experiments**: We present a thorough empirical analysis on real networks to evaluate the summary quality and runtime, and study the properties of seven clustering methods.

For reproducibility, the code will be available online at `https://github.com/yikeliu/ConDeNSe`. Next, we present the related work and necessary background.

Our work is related to graph summarization methods, MDL, and graph clustering. We review each of these topics in turn.

Graph Summarization. Most research efforts focus on plain graphs and can be broadly classified as group-based [LT10, RGM03], compression-based [CKL+09, NRS08], simplification-based, influence-based, and pattern-based [CH94]. Dynamic graph summarization has been studied to a much smaller extent [SKZ+15]. Most related to our work are the ideas of node grouping and graph compression. Built on these ideas, two representative methods, MDL-SUMMARIZATION [NRS08] and VOG [KKVF14], are MDL-based summarization methods that compress the graphs by finding near-structures (e.g., (near-) cliques, (near-) bipartite cores). MDL-SUMMARIZATION, which iteratively combines neighbors into supernodes as long as it helps with minimizing the compression cost, includes mostly cliques and cores in the summaries, and has high runtime complexity. On the other hand, VOG finds structures by employing SLASHBURN [KF11] (explained below) and hence is particularly biased towards stars. Moreover, it creates summaries (i.e., lists of structures) using a greedy heuristic on a pre-ordered set of structures (see below). Unlike these methods, CONDENSE performs ensemble pattern discovery, handles edge-overlapping structures, and its summary assembly is robust to the structure ordering. Thus, it leads to more compact and less biased summaries, creates approximate and easy-to-understand supergraphs, and can be used as a proxy to evaluate clustering methods in a novel way.

MDL in Graph Mining. Many data mining problems are related to summarization and pattern discovery, and, thus, to Kolmogorov complexity [FM07], which can be practically implemented by the MDL principle [Ris83]. Applications include

clustering [CV05], community detection [CPMF04], pattern discovery in static and dynamic networks [KKVF14, SKZ+15], and more.

Graph Clustering. Graph clustering and community detection are of great interest to many domains, including social, biological, and web sciences [GN02, BKM+08, For10]. Here, we leverage several graph clustering methods to obtain diversified graph summaries, since each method is biased toward certain types of structures, such as cliques and bipartite cores [BGLL08, KK99, YL13] or stars [KF11]. Unlike existing literature [LLM10] where clustering methods are compared with respect to classic quality measures, we also propose to use CondeNSe as a vessel to evaluate the methods' summarization power. We leverage seven decomposition methods, which we compare quantitatively in Table 3.1:

• **SlashBurn** [KF11] is a node reordering algorithm initially developed for graph compression. It performs two steps iteratively: (i) It removes high-centrality nodes from the graph; (ii) It reorders nodes such that high-degree nodes are assigned the lowest IDs and nodes from disconnected components get the highest IDs. The process is repeated on the giant connected component. We leverage this process by identifying structures from the egonet of each high-centrality node, and the disconnected components, as subgraphs.

• **Louvain** [BGLL08] is a modularity-based partitioning method for detecting hi-

Table 3.1: Qualitative comparison of the graph clustering techniques included in CondeNSe. Symbols: n = number of nodes, m = number of edges, k = number of clusters/partitions, t = number of iterations, d = average degree, $h(m_h)$ = number of nodes (edges) in hyperbolic structure.

	SlashBurn [KF11]	Louvain [BGLL08]	Spectral [Hes04]	METIS [KK99]	HyCoM [AGMF14]	BigClam [YL13]	KCBC [LSK15]
Overlapping Clusters	✔	✘	✘	✘	✔	✔	✔
Cliques	Many	Many	Many	Many	Some	Many	Many
Stars	Many	Some	Some	Some	Many	Some	Some
Bipartite Cores	Some	Few	Many	Some	Some	Few	Few
Chains	Few	Few	Few	Few	Few	Few	Few
Hyperbolic Structures	Few	Few	Few	Few	Many	Few	Few
Complexity	$O(t(m+n\log n))$	$O(n\log n)$	$O(n^3)$	$O(m\cdot k)$	$O(k(m+h\log h^2 + hm_h))$	$O(d\cdot n\cdot t)$	$O(t(m+n))$
Summarization Power	Excellent	Very Good	Good	Good	Poor	Good	Poor

erarchical community structure. The method is iterative: (i) Each node is placed in its own community. Then, the neighbors j of each node i are considered, and i is moved to j's community if the move produces the maximum modularity gain. The process is applied repeatedly until no further gain is possible. (ii) A new graph is built whose supernodes represent communities, and superedges are weighted by the sum of weights of links between the two communities. The algorithm typically converges in a few passes.

• **Spectral** clustering refers to a class of algorithms that utilize eigendecomposition to identify community structure. We utilize one such spectral clustering algorithm [Hes04], which partitions a graph by performing k-means clustering on the top-k eigenvectors of the input graph. The idea behind this clustering is that nodes with similar connectivity have similar eigen-scores in the top-k vectors and form clusters.

• **METIS** [KK99] is a cut-based k-way multilevel graph partitioning scheme based on multilevel recursive bisection (MLRB). Until the graph size is substantially reduced, it first coarsens the input graph by grouping nodes into supernodes iteratively such that the edge-cut is preserved. Next, the coarsened graph is partitioned using MLRB, and the partitioning is projected onto the original input graph G through backtracking. The method produces k roughly equally-sized partitions.

• **HyCoM** [AGMF14] is a parameter-free algorithm that detects communities with hyperbolic structure. It approximates the optimal solution by iteratively detecting important communities. The key idea is to find in each step a single community that minimizes an MDL-based objective function given the previously detected communities. The iterative procedure consists of three steps: community candidates,

community construction, and matrix deflation.

• **BigClam** [YL13] is a scalable overlapping community detection method. It is built on the observation that overlaps between communities are densely connected. By explicitly modeling the affiliation strength of each node-community pair, the latter is assigned a nonnegative latent factor which represents the degree of membership to the community. Next, the probability of an edge is modeled as a function of the shared community affiliations. The identification of network communities is done by fitting BigClam to a given undirected network G.

• **KCBC** [LSK15] is inspired by the k-cores algorithm [GTV11] that unveils densely connected structures. A k-core is a maximal subgraph for which each node is connected to at least k other nodes. KCBC iteratively removes k-cores starting by setting k equal to the maximum core number (max value k for which the node is present in the resulting subgraph) across all nodes. Each connected component in the induced subgraphs is identified as a cluster, and is removed from the original graph. The process is repeated on the remaining graph.

Other clustering methods that we considered (e.g., Weighted Stochastic Block Model or WSBM) are not included in CondeNSe due to scalability. For instance, WSBM took more than a week to finish on our smallest dataset.

Now we give details of CondeNSe as below.

3.1.1 CondeNSe: Proposed Model

We formulate the graph summarization problem as a graph compression problem. Let $G(\mathcal{V}, \mathcal{E})$ be a graph with $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges, without self-loops. The Minimum Description Length (MDL) problem, which is a practical version of

Table 3.2: Major symbols and definitions.

Notation	Description
$G(\mathcal{V}, \mathcal{E})$, $\mathbf{A}$	graph, and its adjacency matrix
$\mathcal{V}$, $n = \lvert\mathcal{V}\rvert$	node-set and number of nodes of G, resp.
$\mathcal{E}$, $m = \lvert\mathcal{E}\rvert$	edge-set and number of edges of G, resp.
k	# of clusters or communities or patterns
t	# of iterations
h, m_h	size of hyperbolic community, and # of edges in it, resp.
d	average degree of nodes in G
h_{slash}	# of hub nodes to slash per iteration in SlashBurn
fc, bc, st, ch, hs	full clique, bipartite core, star, chain, hyperbolic structure, resp.
$\lvert fc\rvert, \lvert bc\rvert, \lvert st\rvert, \lvert ch\rvert, \lvert hs\rvert$	number of nodes in the corresponding structure
Ω	predefined set of structural pattern types
M	a model or summary for G
s	structure in M
$\lvert S\rvert$, $\lvert s\rvert$	cardinality of set S and number of nodes in s, resp.
$\lvert\lvert s\rvert\rvert, \lvert\lvert s\rvert\rvert'$	# existing and non-existing edges of $\mathbf{A}$ that s describes
$\mathbf{E}$	error matrix, $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$, where $\oplus$ is exclusive OR
$\mathbf{O}$	edge-overlap penalty matrix
$L(G, M)$	# of bits to describe model M, and G using M
$L(M), L(O), L(s)$	# of bits to describe M, the edge overlap O, and structure s

Kolmogorov Complexity [FM07], aims to find the best model M in a given family of models $\mathcal{M}$ for some observed data $\mathcal{D}$ such that it minimizes $L(M) + L(\mathcal{D}|M)$. In this formulation, $L(M)$ is the description length of M in bits and $L(\mathcal{D}|M)$ is the description length of $\mathcal{D}$ which is encoded by the chosen model M. Table 3.2 defines the recurrent symbols used in this section.

We consider summaries in the model family $\mathcal{M}$, which consists of all possible permutations of subsets of structural patterns in Ω. One option is to populate Ω with the frequent patterns that occur in the input graph (in a data-driven manner), but frequent subgraph mining is NP-complete and does not scale well. Moreover, even efficient *approximate* approaches are not applicable to unlabeled graphs and can only handle small graphs with a few tens or hundreds of nodes. To circumvent this problem, we populate Ω with five patterns that are common in real-world static graphs [kle99, AGMF14], correspond to interesting real behaviors, and can (approx-

imately) describe a wide range of structural patterns: stars (*st*), *full* cliques (*fc*), bipartite cores (*bc*), chains (*ch*), and hyperbolic structures with skewed degree distribution (*hs*). Under the MDL principle, any approximate structures (e.g., near-cliques) can be easily encoded as their corresponding exact structures (e.g., *fc*) with some errors. Since many communities have hyperbolic structure [AGMF14], which cannot be expressed as a simple composition of the other structural patterns in Ω, we consider this structure separately. Motivated by real-world discoveries, we focus on structures that are commonly found in networks, but our framework is not restricted to them; it can be readily extended to other, application-dependent types of structures as well.

Formally, we address the following problem:

Problem 3.1. Given a graph G with adjacency matrix $\mathbf{A}$ and structural pattern types Ω, we seek to find the model M that minimizes the encoding length of the graph and the redundancy in edge coverage:

$$L(G, M) = L(M) + L(\mathbf{E}) + L(\mathbf{O}) \tag{3.2}$$

where $\mathbf{M}$ is $\mathbf{A}$'s approximation induced by M, $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$ is the error matrix to correct for edges that were erroneously described by M, $\oplus$ is exclusive OR, and $\mathbf{O}$ is the edge-overlap matrix to penalize edges covered by many patterns.

Model M induces a supergraph with each $s \in M$ as an (approximate) supernode, and weighted superedges between them. Before we further formalize the task of encoding the model, the error matrix, and the edge-overlap penalty matrix, we provide a visual illustration of our MDL objective.

An Illustrative Example. *Figure 3.2 shows the original adjacency matrix* $\mathbf{A}$ *of an input graph, which is encoded as (i)* $\mathbf{M}$*, the matrix that is induced by the model*

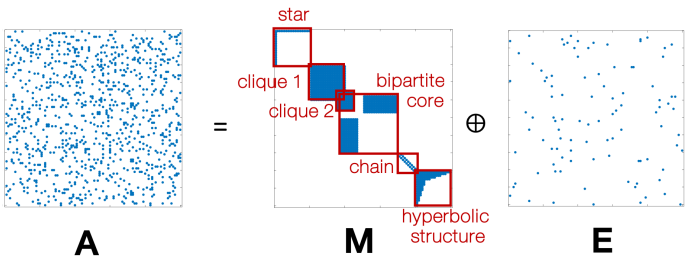


Figure 3.2: Illustration of MDL encoding.

M, and (ii) the error matrix E, which captures additional/missing edges that are not properly described in M. In this example, there are 6 structures in the model (from the top left corner to the bottom right corner: a star, a large clique, a small clique, a bipartite core, a chain, and a hyperbolic structure), where the cliques and the bipartite core have overlapping nodes and edges.

Encoding the Model

Following the literature [KKVF14], for an input graph G, to fully describe a model $M \in \mathcal{M}$ that consists of a set of structural patterns s (e.g., stars, hyperbolic structures, and chains), we encode it as $L(M)$, where we use optimal encoding for all its components:

(3.3)
$$L(M) = L_{\mathbb{N}}(|M| + 1) + \log \binom{|M| + |\Omega| - 1}{|\Omega| - 1} + \sum_{s \in M} \big(- \log_2 \Pr(x(s) \mid M) + L(s) \big).$$

In the first term we encode the number of structural patterns in M using Rissanen's optimal encoding for integers (≥ 1) [Ris83]. Then, we transmit the number of patterns per type in Ω by optimally encoding it via an index over a weak number composition. In the third term, for each structure $s \in M$, we encode its type $x(s)$ using optimal prefix codes [CT12], and its connectivity $L(s)$. For the last term, to capture the real connectivity patterns of each structure s, we introduce the MDL encoding per type of structure in Ω next. As in Equation (3.3), we optimally encode

the various components of each structure (e.g., by using Rissanen's optimal encoding for integers).

- **Stars:** A star consists of a "hub" node connected to two or more "spoke" nodes. We encode it as:

$$L(st) = L_{\mathbb{N}}(|st| - 1) + log_2 n + log_2 \binom{n-1}{|st|-1} \tag{3.4}$$

  where we encode in order the number of spokes, the hub ID (we identify it out of n nodes using an index over the combinatorial number system), and the spoke IDs.

- **Cliques:** A clique is a densely connected set of nodes with:

$$L(fc) = L_{\mathbb{N}}(|fc|) + log_2 \binom{n}{|fc|} \tag{3.5}$$

  where we encode its number of nodes followed by their IDs.

- **Bipartite Cores:** A bipartite core consists of two non-empty sets of nodes, L and R, which have edges only between them, and $L \cap R = \emptyset$. Stars are a special case of bipartite cores with $|L| = 1$. The encoding cost is given as:

$$L(bc) = L_{\mathbb{N}}(|L|) + L_{\mathbb{N}}(|R|) + log_2 \binom{n}{|L|} + log_2 \binom{n}{|R|}, \tag{3.6}$$

  where we encode the number of nodes in L and R followed by the node IDs per set.

- **Chains:** A chain is a series of nodes that are linked consecutively–e.g. node-set $\{a, b, c, d\}$ in which a is connected to b, b is connected to c, and c is connected to d. Its encoding cost, $L(ch)$, is:

$$L(ch) = L_{\mathbb{N}}(|ch| - 1) + \sum_{i=1}^{|ch|} log_2(n - i + 1) \tag{3.7}$$

where we encode its number of nodes, and then their node IDs in order of connection.

- **Hyperbolic Structures:** A hyperbolic structure or community [AGMF14] has skewed degree distribution which often follows a power law with exponent between -0.6 and -1.5. The encoding length of a hyperbolic structure hs is given as:

$$L(hs) = k + L_{\mathbb{N}}(|hs|) + \log_2 \binom{n}{|hs|} + \log_2(|\mathbf{A}(hs)|) + ||hs||l_1 + ||hs||'l_0 \tag{3.8}$$

where we first encode the power-law exponent (using Rissanen's encoding [Ris83] for the integer part, the number of decimal values, and the decimal part), followed by the number of nodes and their IDs. Then, we encode the number of edges in the structure ($=|\mathbf{A}(hs)|$), and use optimal prefix codes, l_0, l_1, for the missing ($||hs||'$) and present ($||hs||$) edges, respectively. Specifically, $l_1 = -\log((||hs||/(||hs|| + ||hs||'))$, and l_0 is defined similarly.

Encoding the Errors

Given that M is a summary, and $\mathbf{M}$ is only an approximation of $\mathbf{A}$, we also need to encode errors of the model. For instance, a near-clique is represented as a full clique in the model, and, thus, contributes some edges to the error matrix (i.e., the missing edges from the real data). We encode the error $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$ in two parts, $\mathbf{E}^+$ and $\mathbf{E}^-$, since they likely follow different distributions [KKVF14]. The former encodes

the edges induced by M which were not in the original graph, and the latter the original edges that are missing in M:

$$L(\mathbf{E}^+) = log_2(|\mathbf{E}^+|) + ||\mathbf{E}^+||l_1 + ||\mathbf{E}^+||'l_0 \tag{3.9}$$

$$L(\mathbf{E}^-) = log_2(|\mathbf{E}^-|) + ||\mathbf{E}^-||l_1 + ||\mathbf{E}^-||'l_0 \tag{3.10}$$

where we encode the number of 1s in $\mathbf{E}^+$ (or $\mathbf{E}^-$), followed by the actual 1s and 0s using optimal prefix codes (as before).

Encoding the Edge-Overlap Penalty

Several of the graph decomposition methods that we consider (e.g., SlashBurn, KCBC in Table 3.1) generate edge-overlapping patterns. The MDL model we have presented so far naturally handles node overlaps—if two structures consist of the same large set of nodes, only one of the them will be chosen during the encoding cost minimization process, because their combination would lead to higher encoding cost. However, up to this point, the model considers a binary state for each edge: that is, an edge is described by the model M, or not described by it. This could lead to summaries with high redundancy in edge coverage, as we show next with an illustrative example.

To explicitly handle extensive edge overlaps in the graph summaries (which can lead to low node/edge coverage), we add a penalty term, $L(\mathbf{O})$, in the optimization function in Equation (3.2). We introduce the matrix $\mathbf{O}$, which maintains the number of times each edge is described by M, i.e., the number of selected structures in which the edge occurs. We encode the description length of $\mathbf{O}$ as:

$$L(\mathbf{O}) = \log_2(|\mathbf{O}|) + ||\mathbf{O}||l_1 + ||\mathbf{O}||'l_0 + \sum_{o \in \mathcal{E}(\mathbf{O})} L_{\mathbb{N}}(|o|) \tag{3.11}$$

where we first encode the number of distinct overlaps, and then use the optimal prefix code to encode the number of the present and missing entries in $\mathbf{O}$. As before, l_0 and l_1 are the lengths of the optimal prefix codes for the present and missing entries, respectively. Finally, we encode the weights in $\mathbf{O}$ using the optimal encoding for integers $L_{\mathbb{N}}$ [Ris83]. We denote with $\mathcal{E}(\mathbf{O})$ the set of non-negative entries in matrix $\mathbf{O}$.

An Illustrative Example. *Let us assume that the output of an edge-overlapping graph clustering method consists of three full cliques: (1) full clique 1 with nodes 1-20; (2) full clique 2 with nodes 11-30; and (3) full clique 3 with nodes 21-40. The encoding that does not account for overlaps (which is based on the modeling described above includes all three structures in the summary, which clearly yields both redundant nodes and edges. Despite the overlap, the description length of the graph given the model above is calculated as* **441** *bits, since edges that are covered multiple times are not penalized. For reference, the graph needs* **652** *bits under the null (empty) model, where all the original edges are captured in the error matrix* $\mathbf{E}$*. Ideally, we want a method that penalizes extensive overlaps and maximizes the node/edge coverage.*

In the example that we described above, by leveraging the full optimization function in Equation (3.2)*, which includes the edge-overlap penalty term, we obtain a summary with only the first two cliques, as desired. The encoding of our proposed method has a length of* **518** *bits, which is higher than the number of bits of the non edge-overlap aware encoding (441 bits). The reason is that in the former (edge-overlap aware) summary, some edges have remained unexplained (edges from nodes 11-20 to nodes 21-40), and thus are encoded as error. On the other hand, the latter summary encodes all the nodes and edges (without errors), but explains many edges twice (e.g.,*

Algorithm 1 CondeNSe

Input: graph G, parameters of clustering methods in Module A
// **Module A: Pattern discovery**: Discovery of a diverse set of patterns P.
P = SlashBurn (G, h_{slash}) $\cup$ Louvain (G, τ) $\cup$ Spectral (G, k) $\cup$ METIS (G, k)

$\cup$ HyCoM (G) $\cup$ BigClam (G) $\cup$ KCBC (G) // discussion of parameters in text

//**Module B:** MDL-based **structural pattern identification** as full cliques, bipartite cores, stars,

// chains and hyperbolic structures.
for $g \in P$
  for $\omega \in \Omega$

    // e.g., hub identification in star structure ω='st'
    $r(g, \omega)$ = 'best' representation of g as structure type ω
  // s: type of structure for pattern g using its best representation $r(g, \omega)$
  $s = \arg\min_{\omega \in \Omega} L_{r(g,\omega)}(g, \omega) = \arg\min_{\omega \in \Omega}\{L(\omega) + L(E^+_\omega) + L(E^-_\omega)\}$ // using Eq. (3.4)-(3.8)
// **Module C: Overlap-aware summary assembly** by employing a Step variant.

$M = \arg\min L(G, M) = \arg\min\{L(M) + L(\mathbf{E}) + L(\mathbf{O})\}$ // Eq. (3.2),(3.3),(3.9)-(3.11)
// **Module D: Approximate supergraph** $G_S(\mathcal{V}_S, \mathcal{E}_S)$ **creation** conditional on the discovered patterns

// (supernodes linked via weighted superedges).

$\mathcal{V}_S = \{s \in M\}$ // supernodes = structures in M

$\mathcal{E}_S = \{(s_i, s_j, w_{ij}) \mid w_{ij} = |\{u, v\}|$, node $u \in s_i$, node $v \in s_j, i \neq j\}$ // superedges
return approximate supergraph $G_S(\mathcal{V}_S, \mathcal{E}_S)$ (summary M)

the nodes 11-20 and the edges between them, the edges between nodes 11-20 and 21-30) without accounting for the redundancy-related bits twice.

Our proposed edge-overlap aware encoding can effectively handle a family model $\mathcal{M}$ that consists of subsets of node- and edge-overlapping structural patterns, and can choose a model M that describes the input graph well, and also minimizes redundant modeling of nodes and edges.

With the model, the errors and the overlapping edges encoded, the CondeNSe algorithm is given as follows.

3.1.2 CondeNSe: Our Proposed Algorithm

Based on the model above, we propose CondeNSe, an ensemble, edge-overlap-aware algorithm that summarizes a graph with a compact supergraph consisting of a diverse set of structural patterns (e.g., *fc*, *hs*). CondeNSe consists of four modules, which we give in Algorithm 1 and describe in detail next.

Module A: Unified Pattern Discovery Module

As we mentioned, earlier, in our formulation, we consider summaries in the model family $\mathcal{M}$, which consists of all possible permutations of subsets of structural patterns in Ω (e.g., a summary with 10 full cliques, 3 bipartite cores, 5 stars and 9 hyperbolic structures). Towards this goal, the first step is to discover subgraphs in the input graph. These can then be used to build its summary. To find the 'perfect' graph summary, we would need to generate all possible (2^n) patterns for a given graph G, and then, from all possible (2^{2^n}) combinations of these patterns pick the set that minimizes Equation (3.2). This is intractable even for small graphs. For example, for $n = 100$ nodes, there are more than $2^{\text{nonillion}}$ (1 nonillion = 10^{30}) possible summaries. We reduce the search space by considering patterns that are found via graph clustering methods, and are likely to fit well the structural patterns in Ω.

The literature is rich in graph clustering methods [BGLL08, KK99, YL13, KF11]. However, each approach is biased towards specific types of structures, which are most often cliques and bipartite cores. Choosing a decomposition method to generate patterns for the summary depends on the domain, the expected patterns (e.g., mainly clique- or star-like structures), and runtime constraints. To mitigate the biases introduced to the summary by individual clustering methods, and consider a diverse set of candidate patterns, we propose a unified approach that leverages seven existing clus-

tering methods: SlashBurn, Louvain, Spectral, METIS, HyCoM, BigClam, and KCBC (which we described above). In Table 3.1, we present the qualitative advantages, disadvantages, and biases of the methods. Specifically, SlashBurn tends to provide excellent graph coverage and biased summaries in which stars dominate. Conversely, most other approaches produce primarily full cliques and stars, and some bipartite cores. HyCoM finds mainly hyperbolic communities with skewed degree distributions.

Our proposed unified approach (Algorithm 1, lines 2-4) is expected to lead to summaries with a better balanced set of structures (i.e., a good mix of exact and approximate cliques, bipartite cores, stars, chains and hyperbolic structures), and lower encoding cost than any standalone graph clustering method. At the same time, it is expected to take longer to generate all the patterns (although the clustering methods can trivially run in parallel), and the search space for the summary becomes larger—equal to the union of all the subgraphs that the clustering methods generate.

In the experimental evaluation, we use CondeNSe to empirically compare the impact of these methods on the summary quality and evaluate their summarization power.

Module B: Structural Pattern Identification Module

This module (Algorithm 1, lines 5-12) identifies and assigns an identifier structural pattern in Ω to all the subgraphs found in module A. In other words, this module seeks to characterize each cluster with its best-suited pattern in $\Omega = \{fc, st, bc, ch, hs\}$. Let g be the induced graph of a pattern generated in Step 1, and ω be a pattern in Ω. Following the reasoning above, we use MDL as a selection criterion. To model g with ω, we first model g with its best representation as structure type ω (explained

in detail next), $r(g, \omega)$, and define its encoding cost as $L_{r(g,\omega}(g, \omega) = L(\omega) + L(g|\omega) = L(\omega) + L(E_\omega^+) + L(E_\omega^-)$, where E_ω^+ and E_ω^- encode the erroneously modeled and unmodeled edges of g. The pattern type in ω that leads to the smallest MDL cost is used as the identifier of the corresponding subgraph g (lines 11-12 in Alg. 1).

Finding the best representation $r(g, \omega)$. Per pattern type ω, each pattern g can be represented by a family of structures—e.g., we can represent g with as many bipartite cores as can be induced on all possible permutations of g's nodes into two sets L (left nodeset) and R (right nodeset) . The only exception is the full clique (fc) pattern, which has a unique (unordered) set of nodes. To make the problem tractable, we use the graph-theoretical properties of the pattern types in Ω in order to choose the representation of g which minimizes the incorrectly modeled edges.

Specifically, we represent g as a star by identifying its highest-degree node as the hub and all other nodes as spokes. Representing g as a bipartite core reduces to finding the maximum bipartite pattern, which is NP-hard. To scale-up the computation, we approximate it with semi-supervised classification with two classes L and R, and the prior information that the highest-degree node belongs to L and its neighbors to R. For the classification, we use Fast Belief Propagation [KKK$^+$11] with heterophily between neighbors. Similarly, representing g as a chain reduces to finding its longest path, which is also NP-hard. By starting from a random node, we perform Breadth First Search two times, and end on nodes v_1 and v_2, respectively. Then, we consider the path v_1 to v_2 (based on BFS), and perform local search to further expand it. For the hyperbolic structures, we used power-law fitting (`http://tuvalu.santafe.edu/\~aaronc/powerlaws/` by Clauset et al.). Lines 7-10 in Algorithm 1 succinctly describe the search of the best representation r for every subgraph g and pattern

type ω.

Module C: Structural Pattern Selection Module

This module is key for creating *compact* summaries and is described in lines 13-14 of Alg. 1. Ideally, we would consider all possible combinations of the previously identified structures and pick the subset that minimizes the encoding cost in Equation (3.2). If $|\mathcal{S}|$ structures have been found and identified in the previous steps, finding the optimal summary from $2^{|\mathcal{S}|}$ possibilities is not tractable. For reference, we have seen empirically that graphs with about 100,000 nodes, have over $50K$ structures. The optimization function is neither monotonic nor submodular, in which case a greedy hill climbing approach would give a $(1 - \frac{1}{\epsilon})$-approximation of the optimal.

Instead of considering all possible combinations of structures for the summary, prior work has proposed GnF, a heuristic that considers the structures in decreasing order of "local" encoding benefit and includes in the model the ones that help further decrease the graph's encoding cost $L(G, M)$. The local encoding benefit [KKVF14] is defined as $L(g, \emptyset) - L(g, \omega)$, where $L(g, \emptyset)$ represents the encoding of g as noise (i.e., empty model). Although it is efficient, its output summary and performance heavily depend on the structure order. To overcome these shortcomings and obtain more compact summaries, we propose a new structural pattern selection method, Step, as well as a faster serial version and three parallel variants: Step-P, Step-PA, and k-Step.

• **Step.** This method iteratively sifts through all the structures in $\mathcal{S}$ and includes in the summary the structure that decreases the cost in Equation (3.2) the most, until no structure further decreases the cost. Formally, if $\mathcal{S}_i$ is the set of structures that have not been included in the summary at iteration i, Step chooses structure s_i^* s.t.

$$s_i^* = arg\min_{s \in \mathcal{S}_i} L(G, M_{i-1} \cup \{s\})$$

where M_{i-1} is the model at iteration $i-1$, and $M_0 = \emptyset$ is the empty model. CONDENSE with STEP finds up to 30% more compact summaries than baseline methods, but its quadratic runtime $O(|\mathcal{S}|^2)$ makes it less ideal for large datasets with many structures $\mathcal{S}$ produced by module A. Therefore, we propose four methods that significantly reduce STEP's runtime while maintaining its summary quality.

• **Step-P.** The goal of STEP-P is to speed up the computation of STEP by iteratively solving smaller, "local" versions of STEP in parallel. STEP-P begins by dividing the nodes of the graph into p partitions using METIS. Next, each candidate structural pattern is assigned to the partition with the maximal node overlap. STEP-P then iterates until convergence, with each iteration consisting of two phases:

1. **Parallelize.** In parallel, a process is spawned for each partition and is tasked with finding the structure that would lower the encoding cost in Eq. (3.2) the most out of all the structures in its partition. For any given partition, there may be no structure that lowers the global encoding cost.

2. **Sync.** From all structures returned in phase 1, the one that minimizes Equation (3.2) the most is added to the summary. If no structure reduces the encoding cost, the algorithm has converged. If not, phase 1 is repeated.

• **Step-PA.** In addition to parallelizing STEP, we introduce the idea of "inactive" partitions, which is an optimization designed to reduce the number of processes that are spawned by STEP-P. STEP-PA differs from STEP-P by designating every partition of the graph as active, then if a partition fails x times to find a structure

that lowers the cost in Equation (3.2), that partition is declared inactive and is not visited in future iterations. Thus, the partitions with structures not likely to decrease the overall encoding cost of the model get x chances (e.g. 3) before being eventually ruled out, effectively reducing the number of processes spawned for each iteration of STEP-PA after the first x iterations.

• **k-Step.** The pseudocode of this variant is given in Algorithm 2. K-STEP further speeds up STEP while maintaining high-quality summaries. This algorithm has two phases: the first applies STEP-P K times (lines 3-5) to guarantee that the initial structural patterns included in the summary are of good quality. The second expands the summary by building local solutions of STEP-P per active partition (lines 8-9). If a partition does not return any solution, it is flagged as inactive (lines 10-11). For the partitions that returned non-empty solutions, the best structure per partition is added into a temporary list (line 13), and a parallel "glocal" step applies STEP-P over that list and populates the summary (lines 14-16). We refer to this step as "glocal" because it is a global step within the local stage. The local stage is repeated until no active partitions are left.

Module D: Approximate Supergraph Creation Module

In the empirical analysis (see following), we show that STEP results in graph summaries with up to 80-90% fewer structures than the baselines, and thus can be leveraged for tractable graph visualization. The last and fourth module of CONDENSE (Algorithm 1, lines 15-18), instead of merely outputting a list of structures, creates an "approximate" supergraph which gives a high-level but informative view of large graphs. An *exact* supergraph, $G_S(\mathcal{V}_S, \mathcal{E}_S)$, of a graph $G(\mathcal{V}, \mathcal{E})$ consists of a set of supernodes $\mathcal{V}_S = P(\mathcal{V})$ which is a power set (i.e., family of sets) over $\mathcal{V}$ and a set of

Algorithm 2 K-STEP

```
1: Input: graph G(V, E); list of structures S; P partitions; K iterations
2: ActivePartitions = {1, ..., P}                          // all partitions are active
3: // Stage 1: Global
4: for i = 1 : K
5:
6:     run STEP-P ()                                       // summary of K structures
7: // Stage 2: Local Stage
8: repeat:
9:     for p ∈ ActivePartitions:                           // 2.1: Local sub-stage
10:        s = run STEP-P-Parallelize()                    // s = best structure in p
11:        if s = ∅                                        // no structure returned
12:            ActivePartitions.remove(p)                  // partition p is inactive
13:        else
14:            bestStructs.add(s)                          // s is candidate for M
15:                                                        // p remains active
16:    repeat:                                             // in parallel, add structures to M
17:        run STEP-P-Sync(bestStructs)                    // 2.2: Glocal sub-stage
18:    until bestStructs = ∅ or Eq. (3.2) is minimized
19: until ActivePartitions = ∅ return M
```

superedges $\mathcal{E}_S$. The superweight is often defined as the sum of edge weights between the supernodes' constituent nodes.

Unlike most prior work, CONDENSE creates "approximate," yet powerful supergraphs: (i) the supernodes do not necessarily correspond to a set of nodes with the same connectivity, but to *rich* structural patterns (including hyperbolic structures and chains); (ii) the supernodes may have *node overlap*, which helps to pinpoint bridge nodes (i.e., nodes that span multiple communities); (iii) the supernodes may show deviations from the perfect corresponding structural patterns (i.e., they correspond to near-structures).

Definition 3.12. A **CondeNSe approximate supergraph** of G is a supergraph with supernodes that correspond to *possibly-overlapping structural patterns* in Ω. These patterns are approximations of clusters in G.

In other words, the CONDENSE supergraphs consist of supernodes that are *fc*, *st*, *ch*, *bc*, and *hs*. To obtain an approximate supergraph, we map the structural patterns returned in module C to approximate supernodes. Then, for every pair of supernodes,

we add a superedge if there were edges between their constituent nodes in $\mathcal{V}$ and set its superweight equal to the number of such (unweighted) edges, as shown in line 18 of Algorithm 1.

To evaluate the edge overlap in the summaries, and hence the effectiveness of our overlap-aware encoding, we use the normalized overlap metric. The normalized overlap between two supernodes is their Jaccard similarity. It is 0 if the supernodes do not share any nodes, and close to 1 if they share many nodes compared to their sizes. Although it is not the focus of the current paper, the CondeNSe supergraphs can be used for visualization and potentially for approximation of algorithms on large networks (without specific theoretical guarantees, at least in the general form).

CondeNSe: Complexity Analysis

We discuss the complexity of CondeNSe by considering each module separately:

The first module has complexity $O(n^3)$, which corresponds to Spectral. However, in practice, HyCoM is often slower than Spectral, likely due to implementation differences (JAVA vs. MATLAB). The complexity of this module can be lowered by selecting the fastest methods. Module B is linear on the number of edges of the discovered patterns. Given that they are overlapping, the computation of $L(G, M)$ is done in $T = O(|M|^2+m)$, which is $O(m)$ for real graphs with $|M|^2 << m$. In module C, Step has complexity $O(|S|^2 \times T)$, where S is the set of labeled structures. Step-P and Step-PA are $O(t \times \frac{|S|^2}{p} \times T)$, where p is the number of METIS partitions ('active' partitions for Step-PA) and t is the number of iterations. k-Step is a combination of Step-P and a local stage, so it runs in $O(K \times \frac{|S|^2}{p} \times T + t_{lcl} \times (\frac{|S|^2}{p_{active}} + p^2_{active}) \times T)$, where t_{lcl} is the iterations of its local stage. Finally, the supergraph (module D) can be generated in $O(m)$.

Table 3.3: Summary of graphs used in our experiments.

Name	Nodes	Edges	Description
`EUmail` [LK14]	265,214	420,045	EU uni. email comm.
`Enron` [LK14]	80,163	288,364	Enron email comm.
`AS-Caida` [LK14]	26,475	106,762	BGP routing table
`AS-Oregon` [LK14]	13,579	37,448	Router connections
`Choc`	2,899	5,467	Co-editor wiki graph

3.1.3 Empirical Analysis

We conduct thorough experimental analysis to answer three main questions:

- How effective is CondeNSe?
- Does it scale with the size of the input graph?
- How do the clustering methods compare in terms of summarization power?

Setup. We ran experiments on the real graphs given in Table 3.3. As far as the parameter setting for the clustering methods is concerned, for SlashBurn, we choose the number of hub nodes to slash per iteration $h_{slash} = 2$ in order to achieve better granularity of clusters. For Louvain, we choose resolution $\tau = 0.0001$ as it generates comparable number of clusters with other clustering methods for all our datasets. For Spectral and METIS, the number of clusters k are set to $\sqrt{n/2}$ according to a rule of thumb [cyt], where n is the number of nodes in the graph. As for other clustering methods, they are parameter-free hence no need to set up parameters. Unless otherwise specified, we followed the same rule of thumb for setting the number of input METIS partitions p for all the Step variants. In the following experiments, we set the number of chances $x = 3$ for Step-PA.

Effectiveness of CondeNSe

Ideally, we want a summary to be: (i) concise, with a small number of struc-

tures/supernodes; (ii) minimally redundant, i.e., capturing dependencies such as *overlapping* supernodes, but without overly encoding overlaps; and (iii) covering in terms of nodes and edges. Our proposed method, CondeNSe, constitutes an (almost) unbiased way of analyzing the structure of a given graph. How does it fare in terms of these properties? To answer the question, we perform experiments on the real data in Table 3.3.

Baselines. The first baseline is VoG [KKVF14], which we describe above. For our experiments, we used the code that is online at `https://github.com/GemsLab/VoG_Graph_Summarization`. The second baseline is our proposed method, CondeNSe, combined with the GnF heuristic (described above).

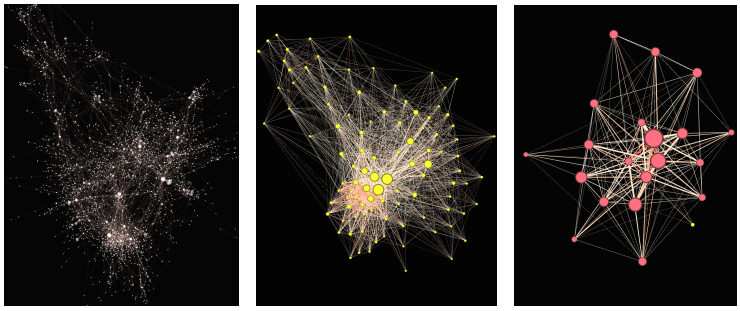
A1. Conciseness. In Table 3.4, we compare our proposed method (for different selection methods) and the baselines with respect to their compression rates, i.e., the percentage of bits needed to encode a graph with the composed summary over the number of bits needed to encode the corresponding graph with an empty model/summary (that is, all the edges are in the error matrix). In parentheses, we also give the total number of structures in the summaries. We see that compared to the baselines, CondeNSe with the Step variants gives significantly more compact summaries, with 30%-50% lower compression rate and about 80-90% fewer structures. The Step variants give comparable results in summarization power.

Table 3.4: CondeNSe: Compression rate with respect to the empty model. In parentheses, number of structures in the corresponding summary. A "-" means that the corresponding method was terminated after 4 days. (* In the interest of time, the summary size was limited to 15.)

Dataset	**VoG [KKVF14]**	**CondeNSe** GnF	**CondeNSe with Step Variants**			
			Step	**Step-P**	**Step-PA**	**k-Step**
`Choc`	88%(101)	88%(101)	56%(24)	56%(24)	56%(21)	56%(22)
`AS-Oregon`	71%(400)	69%(379)	35%(41)	35%(41)	35%(35)	35%(36)
`AS-Caida`	-	71%(572)	42%(51)	42%(51)	42%(46)	44%(60)
`Enron`	75%(2330)	74%(2044)	-	26%(50)	25%(201)	25%(218)
`EUmail`	-	65%(1440)	-	-	-	59%(15*)

Table 3.5: Overlapping supernode pairs and average similarity in parentheses. CondeNSe reduces the overlap. A "-" means that the corresponding method was terminated after 4 days.

Dataset	VoG [KKVF14]	CondeNSe
Choc	900 (0.04)	74 (0.029)
AS-Oregon	15875 (0.047)	126 (0.026)
AS-Caida	-	382 (0.018)
Enron	447052 (0.02)	509 (0.015)
EUmail	-	0



(a) Original graph (b) VoG [KKVF14] (c) CondeNSe-Step

Figure 3.3: Choc: CondeNSe-Step generates more compact supergraphs. (b)-(c): The full supergraphs by VoG-GnF, and CondeNSe-Step, resp. Yellow for stars, red for cliques, green for bipartite cores. The edge weights correspond to the number of inter-supernode edges.

A2. Minimal Redundancy. In Figures 3.1 and 3.3, we visualize the supergraphs for AS-Oregon and Choc, which are generated from the selected structures of VoG and CondeNSe-Step. It is clear that the CondeNSe supergraphs are significantly more compact. In Table 3.5, we also provide information about the number of overlapping supernode pairs and their average Jaccard similarity, as an overlap quantifier (in parentheses). For brevity, we only give results for k-Step, since the results of the rest Step-series are similar. We observe that CondeNSe has significantly fewer supernode overlaps, and the overlaps are smaller in magnitude. We also note that the overlap encoding module achieves 10-20% reduction in overlapping edges, showing its effectiveness for minimizing redundancy.

A3. Coverage. We give the summary node/edge coverage (as a ratio of the original) for different assembly methods in Figure 3.4. We observe that the baselines have better edge coverage than the Step variants, which is expected as they include

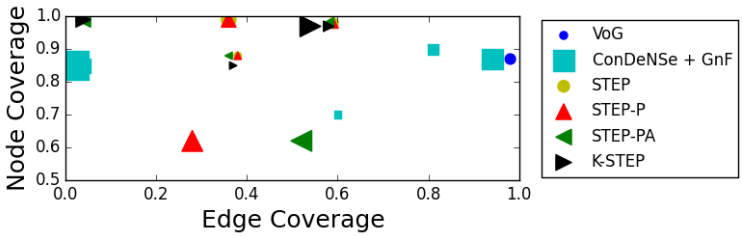


Figure 3.4: Node coverage vs. edge coverage—marker size corresponds to the graph size. STEP variants have better node coverage, and handle the summary coverage-conciseness trade-off well.

significantly more structures in their summaries. However, in most cases, K-STEP and STEP-PA achieve better node coverage than the baselines. Taking into account the (contradicting) desired property for summary conciseness, CONDENSE with STEP variants has better performance, balancing coverage and summary size well.

What other properties do the various summaries have? What are the main structures found in different types of networks (e.g., email vs. routing networks)? In Table 3.6, we show the number of in-summary structures per type. We note that no chains and hyperbolic structures were included in the summaries of the networks that we show here (although some were found by the pattern discovery module, and there are synthetic examples in which they are included in the final summaries). This is possibly because stars are extreme cases of hyperbolic structures, and the encoding of (approximate) hyperbolic structures is of the same order, yet often more expensive than the encoding of stars with errors. As for chains, they are not 'typical' clusters found by popular clustering methods, but rather by-products of the decomposition methods that we consider. Moreover, given that the chain encoding considers the sequence of node IDs, and errors in the real data increase the encoding cost, very often encoding them in the error matrix yields better compression. One observation is that STEP gives less biased summaries than the baselines. For email networks, we see that stars are dominant (e.g., users emailing multiple employees that do not contact

Table 3.6: CondeNSe: Number of structures per type in the summaries in the format $[fc, st, bc]$, for VoG we have $[fc + nc, st, bc + nb]$, where nc is near-clique and nb is near-bipartite core. The CondeNSe summaries are more balanced, *without* a specific pattern type dominating in all the graphs. In the interest of time, we find the top-50 and top-15 structures for `Enron` and `EUmail`, respectively. A "-" means that the corresponding method was terminated after 4 days.

Dataset	VoG [KKVF14]	CondeNSe-GnF	CondeNSe with Step Variants			
			Step	Step-P	Step-PA	k-Step
`Choc`	[0,101,0]	[1,100,0]	[21,3,0]	[21,3,0]	[20,1,0]	[21,1,0]
`AS-Oregon`	[1,399,0,]	[19,355,5]	[27,13,1]	[27,13,1]	[26,9,0]	[26,10,0]
`AS-Caida`	-	[2,557,13]	[38,7,6]	[38,7,6]	[37,5,4]	[43,12,5]
`Enron`	[2,2323,5]	[160,1676,208]	-	[45,2,3]	[60,108,33]	[61,124,33]
`EUmail`	-	[0,1261,179]	-	-	-	[15,0,0]

each other), with considerable number of cliques and bipartite cores too. For routing networks (`AS-Caida` and `AS-Oregon`), we mostly see cliques (e.g., "hot-potato" routing), and a few stars and bipartite cores. In collaboration networks, cliques are the most common structures, followed by stars (e.g., administrators). VoG and CondeNSe-GnF are biased towards stars, which exceed the other structures by an order of magnitude. Overall, CondeNSe fares well with respect to the desired properties for graph summaries.

Runtime Analysis of CondeNSe

We give the runtime of pattern discovery and the Step methods in Figure 3.5. "Discovery" represents the maximum time of the clustering methods, and "Disc.-Fast" corresponds to the slowest among the fastest methods (KCBC, Louvain, METIS, BigClam). We ran the experiment on an Intel(R) Xeon(R) CPU E5-1650 at 3.50GHz, and 256GB memory.

We see that the fast unified discovery is up to 80× faster than the original one. As expected, Step is the slowest method. The parallel variants Step-P, Step-PA, and k-Step are more scalable, with k-Step being the most efficient. Taking into account the similarity of the heuristics in both conciseness and coverage, Figure 3.5 further suggests that k-Step is the best-performing heuristic given that it exhibits

the shortest runtime.

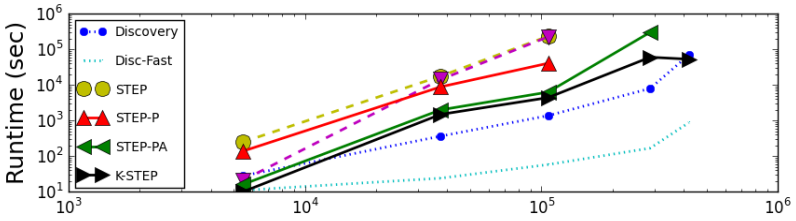


Figure 3.5: Runtime vs. # of edges: K-STEP is more efficient than the other methods, and scales to larger graphs.

Sensitivity Analysis of CondeNSe: Agreement between Step and Step-variants

Our analysis so far has shown that K-STEP leads to the best combination of high compression and low runtime compared to the other methods. But how well does it approximate STEP in terms of the generated summary? To answer this question, we evaluate the "agreement" between the generated summaries, which in this section we view as ordered lists of structures based on the iteration they were included in the final summary (which defines the rank of each structure). Since popular rank correlation measures, such as Spearman's ρ, Kendall's τ, only work on permuted lists or lists of the same length, while the generated summaries can have different constituent structures and lengths, we propose AG as a measure of agreement. This measure effectively handles summaries of different lengths, and penalizes with different, adaptive weights 'rank' disagreement between structures included in both summaries, and disagreement for missing structures from one summary. Let M_1 and M_2 be the two summaries, and $rank(s, M_i)$ be the ranking of structure s in summary M_i (i.e., the order in which it was included in the summary while minimizing Eq. (3.2)). We define the agreement of the two summaries as:

Table 3.7: Agreement of STEP and its variants. They approximate STEP quite well. (* Agreement based on the top-50 structures due to STEP-P's lack of scalability.)

Dataset	Step-P	Step-PA	k-Step
Choc	1	0.9886	0.9667
AS-Oregon	1	0.9704	0.9285
AS-Caida	1	0.9865	0.8238
Enron	1	0.5012*	0.446*

$$AG(M_1, M_2) = 1 - 1/Z[\alpha D + (1 - \alpha/2)D_1 + (1 - \alpha/2)D_2]$$

where $D = \sum_{s \in M_1 \cap M_2} |rank(s, M_1) - rank(s, M_2)|$ is the rank disagreement for structures that are in both summaries, $D_1 = \sum_{s \in M_1 \cap M_2'} |(|M_2| + 1) - rank(s, M_1)|$ is the disagreement for structures in M_1 but not in M_2, D_2 is defined analogously to capture structures in M_2 but not in M_1. Finally, Z is a normalization factor that guarantees that AG is in $[0, 1]$: $Z = (1 - \frac{\alpha}{2}) \sum_{s \in M_1} |(|M_2| + 1) - rank(s, M_1)| + \sum_{s \in M_2} |(|M_1| + 1) - rank(s, M_2)|$. $AG = 1$ means identical summaries, while 0 completely different summaries. In order to penalize more the structures that appear in one summary but not in the other, we set $\alpha = 0.3$ (the results are consistent for other values of α). In Table 3.7, we give the agreement between STEP and its faster variants. As a side note, the agreement with VOG is almost 0 in all the cases. As expected, STEP-P produces the same summaries as STEP, while STEP-PA and K-STEP preserve the agreement well.

Sensitivity to the number of partitions All parallel variants of STEP take p METIS partitions as input. To analyze the effects of varying p on runtime and agreement, we ran K-STEP and increased p from 12 to 96 in increments of 12.

We only give the results on AS-Oregon, since other datasets lead to similar results. We observe that while agreement is robust, runtime decreases as p increases and especially so with the smaller values of p. This observation is consistent with our

motivation for parallelizing STEP: by decreasing the number of structures in any given partition, the "local" subproblems of STEP become smaller and thus less time-consuming. Figure 3.6a shows the effect of the number of partitions on runtime and agreement, both averaged over three trials.

Sensitivity of Step-PA

We also experimented with varying the number of "chances" allowed for partitions in the STEP-PA variant. STEP-PA speeds up STEP-P by forcing partitions to drop out after not returning structures for a certain number of attempts (x). However, while giving partitions fewer chances can speed up the algorithm, smaller values of x can compromise compression and agreement.

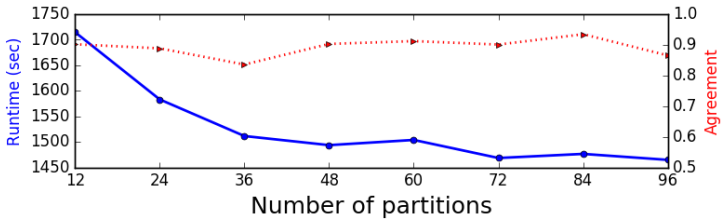
In Figure 3.6b, we give the agreement and runtime of STEP-PA on `Choc` and `AS-Oregon` setting $x = \{1, 2, 3, 4, 5\}$. We found that both runtime and agreement increased with x, and plateaued after $x = 3$. This suggests that forcing partitions to drop out early, while better for runtime, can lead to the loss of candidate structures that may be useful for compression later.

CondeNSe as a Clustering Evaluation Metric

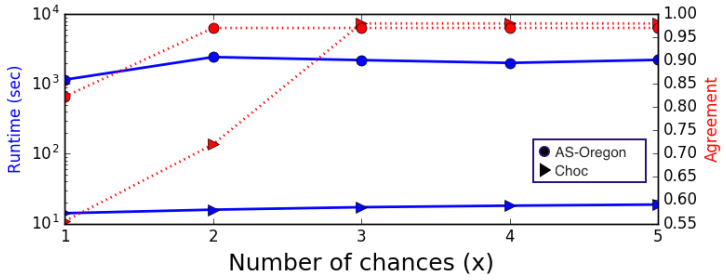
Given the independence of STEP from the structure ordering, we use CONDENSE to evaluate the different clustering methods and give their individual compression rates in Table 3.8.

Table 3.8: CONDENSE as an evaluation metric: Compression rate of clustering methods with respect to the empty model (i.e., percentage of bits for encoding the graph given the chosen model vs. the empty model).

Dataset	Clustering Methods						
	SlashBurn	**Louvain**	**Spectral**	**METIS**	**HyCoM**	**BigClam**	**KCBC**
`Choc`	88%	99%	99%	100%	100%	87%	**78%**
`AS-Oregon`	76%	94%	82%	85%	98%	83%	**65%**
`AS-Caida`	**70%**	100%	100%	98%	98%	91%	74%



(a) `AS-Oregon`: Runtime and agreement vs. number of partitions.



(b) `Choc` + `AS-Oregon`: Runtime and agreement vs. number of chances (x).

Figure 3.6: The agreement is robust to the number of partitions, while the runtime decreases.

For number and type of structures we give our observations based on `AS-Oregon` (Figure 3.7), which is consistent with other datasets. As we see in the case of `AS-Oregon` (which is consistent with the other networks), SlashBurn mainly finds stars (136 out of 138 structures); Louvain, Spectral, KCBC, and BigClam reveal mostly cliques (9/9, 15/17, 9/9, and 28/29, respectively); METIS has a less biased distribution (18 cliques, 12 stars), and HyCoM, though looks for hyperbolic structures, tends to find cliques in our experiments (45 out of 52 structures). Also, SlashBurn and BigClam discover more structural patterns than other methods, which partially explains their good compression rate in Table 3.8.

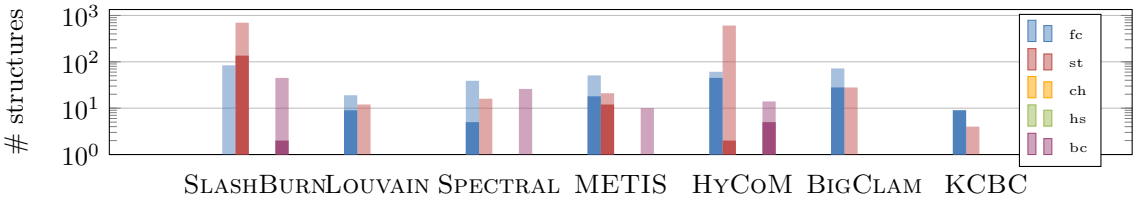


Figure 3.7: Number of structures found by the clustering methods for `AS-Oregon`. Transparent/solid rectangles for before/after the structure selection step. Notation: fc: full clique, st: star, ch: chain, bc: bipartite core, hs: hyperbolic structure.

Table 3.9: Ablation study for `AS-Oregon`. LOUVAIN and SLASHBURN contribute most to the CONDENSE summaries.

Clustering Method	Compression Rate	Contribution per Method						
		SlashBurn	**Louvain**	**Spectral**	**METIS**	**HyCoM**	**BigClam**	**KCBC**
SlashBurn	22%	-	**63%**	10%	7%	7%	0	13%
Louvain	**30%**	30%	-	16%	**45%**	0	3%	7%
Spectral	22%	32%	**51%**	-	3%	0	0	14%
METIS	22%	34%	**46%**	5%	-	2%	0	12%
HyCoM	22%	35%	**48%**	3%	3%	-	0	13%
BigClam	22%	34%	**46%**	2%	2%	2%	-	12%
KCBC	25%	**50%**	35%	6%	2%	2%	6%	-

We perform an ablation study to evaluate the graph clustering methods in the context of summarization. Specifically, we create a leave-one-out unified model for each clustering method and evaluate the contribution of each clustering method to the final summary. The results are shown in Table 3.9. We see that LOUVAIN appears to be the most important method: when included, it contributes the most; and when dropped, the compression rate reduces (worse). When KCBC is dropped, SLASHBURN gets to the top, but LOUVAIN also has considerable contribution. In the missing-LOUVAIN case, the contribution gets redistributed among other clustering methods to make up for it, this effect differs by dataset, e.g., METIS gets boosted for `AS-Oregon`, while it is SPECTRAL for `Choc`.

In terms of runtime, for modules A and B (pattern discovery and identification), SPECTRAL and HYCOM take the longest time, while KCBC, LOUVAIN, METIS, and BIGCLAM are the fastest ones, with SLASHBURN falling in the middle. For Module C (summary assembly), the trade-off between runtime and candidate structures is given in the complexity analysis (see above). In practice, HYCOM usually takes the longest time, followed by SPECTRAL and SLASHBURN.

Conclusion

In this work we proposed CONDENSE, a method that summarizes large graphs as small, approximate and high-quality supergraphs conditioned on diverse pattern

types. CondeNSe features a new selection method, Step, which generates summaries with high compression and node coverage. However, this comes at the cost of increased runtime, which we addressed by introducing faster parallel approximations to Step. We provided a thorough empirical analysis of CondeNSe, and contributed a novel evaluation of clustering methods in terms of summarization power, complementing the literature that focuses on classic quality measures. We showed that each clustering approach has its strengths and weaknesses and make different contributions to the final summary. Moreover, CondeNSe leverages their strengths, handles edge-overlapping structures, and shows results superior to baselines, including significant improvement in the bias of summaries with respect to the considered pattern types.

Ideally without the constraint of time, we naturally recommend the application of as many clustering methods in Module A of CondeNSe. On the other hand, to deal with the additional complexity of having more structures, we recommend choosing faster clustering methods or a mixture of fast and 'useful' methods (depending on the application at hand) that contribute good structures, as shown in our analysis.

3.2 Network Clustering

In CondeNSe, network clustering is heavily used, both serving as a module for CondeNSe and also being evaluated by CondeNSe for its summarization power. In fact, in graph mining, clustering and summarization are connected more tightly than most readers presume. I will show some major works in network clustering and how they are related to network summarization, more details can be found in [LSDK18].

In [LSDK18], we propose a taxonomy for graph summarization. The most popular type of algorithm fall into the category of grouping- or aggregation-based methods, wehre Some *node-grouping* methods recursively aggregate nodes into "supernodes" based on an application-dependent optimization function, which can be based on structure and/or attributes. Others employ existing clustering techniques and map each densely-connected cluster to a supernode. *Edge-grouping* methods aggregate edges into compressor or virtual nodes. I only talk about node-grouping methods as they essentially clustering methods.

Some summarization approaches employ existing clustering techniques to find clusters that then map to supernodes. Others recursively aggregate nodes into supernodes, connected via superedges, based on an application-dependent optimization function.

Node clustering-based methods. Although node grouping and clustering are related in that they result in collections of nodes, they have different goals. In the context of summarization, node grouping is performed so that the resultant graph summary has specific properties, e.g., query-specific properties or maintenance of edge weights. On the other hand, clustering or partitioning usually targets the minimization of cross-cluster edges or a variant thereof, without the end goal of producing a graph summary. Moreover, unlike role mining [HGL$^+$11, HGER$^+$12, GERD13] or structural equivalence [PS89], which seek to identify "functions" of nodes (e.g., bridge or spoke nodes) and find role memberships, summarization methods seek to group nodes that have not only structural similarities, but are also connected or close to each other in the network and thus can be replaced with a supernode.

Although the goal of clustering is not graph summarization, the outputs of clustering algorithms can be easily converted to non-application-specific summaries. In a

nutshell, a small representation of the input graph can be obtained by (i) mapping all the nodes that belong to the same cluster / community to a supernode, and (ii) linking them with superedges with weight equal to the sum of the cross-cluster edges, or else the sum of the weights of the original edges [NG04, YL13, LBG+12]. Although the clustering output can be viewed as a summary graph, a fundamental difference from tailored summarization techniques is that the latter groups nodes that are linked to the rest of the graph in a similar way, while clustering methods simply group densely-connected nodes. There exist comprehensive introductions to clustering techniques [LRU14, Agg15] and work on clustering or community detection methods [AW10], so we do not cover them in this survey. Among the most popular partitioning methods are Graclus [DGK05], spectral partitioning [AKY99], and METIS [KK99]. Although METIS is a well-known partitioning approach that finds "hard" node memberships, it constructs a series of graph "summaries" by iteratively finding the maximal graph matching and merging nodes that are incident to an edge of the matching. The bisection result on the most coarsened graph is then projected backwards to the original graph. Via this process, it is possible to obtain a compact, hierarchical representation of the original graph, which resembles other node-grouping summarization methods.

Node aggregation-based methods. One representative algorithm of hierarchical and clustering-based node grouping is GraSS [LT10], which targets accurate query handling. This summarization method supports queries on the adjacency between two nodes, as well as the degree and the eigenvector centrality of a node. The graph summaries are generated by greedily grouping nodes such that the normalized reconstructed error, $\frac{1}{|\mathcal{V}|^2}\sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{V}}|\bar{A}(i,j)-A(i,j)|$, is minimized—$\mathbf{A}$ is the original adjacency matrix of the graph and $\bar{\mathbf{A}}$ is the real-valued approximate adjacency ma-

trix, each entry of which intuitively represents the probability of the corresponding edge existing in the original graph given the summary. The resulting summaries are represented as a group of vertex sets with information about the number of edges within and between clusters. These sets are used to generate a probabilistic approximate adjacency matrix upon which incoming queries are computed. For example, if many edges cross vertex sets A and B, then it is likely that a node in A is connected to a node in B. In another variant, GraSS leverages Minimum Description Length (MDL) to automatically find the optimal number of supernodes in the summary.

While GraSS does not guarantee output quality, [RGSB14] propose a method of generating supernodes and superedges with guarantees. Here, the objective is to find a supergraph that minimizes the l_p-reconstruction error, or the p-norm of $\mathbf{A} - \bar{\mathbf{A}}$, as opposed to the normalized reconstruction error in GraSS, given a number of supernodes k. The proposed approach, which uses sketching, sampling, and approximate partitioning, is the first polynomial-time approximation algorithm of its kind with runtime $O(|\mathcal{E}| + |\mathcal{V}| \cdot k)$. This method targets efficiency for the same types of queries as GraSS, as well as triangle and subgraph counting queries.

[TZHH11] focus on compressing graphs with edge weights, proposing to merge nodes with similar relationships to other entities (structurally equivalent nodes) such that approximation error is minimized and compression is maximized. In merging nodes to obtain a compressed graph, the algorithm maintains either edge weights or strengths of connections of up to a certain number of hops. Specifically, in the simplest version of the solution, each superedge is assigned the mean weight of all edges it represents. In the generalized version, the best path between any two nodes is "approximately equally good" in the compressed graph and original graphs, but the paths do not

have to be the same. The definition of path "goodness" is data- and application-dependent. For example, the path quality can be defined as the maximum flow through the path for a flow graph, or the probability that the path exists for a probabilistic or uncertain graph.

The methods described above all minimize some version of the approximation or reconstruction error. Other node-grouping approaches seek summaries that maintain specific properties of the original graph, a goal that resembles the target of graph sparsification methods [SS11, HKBG08]. One example is diffusive properties related to the spectrum of the graph, and specifically its first eigenvalue λ_1 [PPK$^+$14], which are crucial in diffusion and propagation processes like epidemiology and viral marketing. In this case, the summarization problem is formulated as a minimization of the change in the first eigenvalue between the adjacency matrices of the summary and the original graph. For efficiency, the method repeatedly merges pairs of *adjacent* nodes, and uses a closed form to evaluate the change in λ_1, derived using matrix perturbation theory. Node pairs are merged in increasing order of change in λ_1—the light edges with small "edge scores" in step 1 are good candidates for merging—and the merging process stops when the user-specified number of nodes is achieved. At every step, edges are reweighted so that λ_1 is maintained (see [LSDK18] for more details). The temporal extension of this approach is also discussed in [LSDK18].

In the visualization domain, [DS13] introduce motif simplification to enhance network visualization. Motif simplification replaces common links and common subgraphs, like stars and cliques, with compact glyphs to help visualize and simplify the complex relationships between entities and attributes. This approach uses exact pattern discovery algorithms to identify patterns and subgraphs, replacing these with glyphs

to result in a less cluttered network display.

Beyond the end goal of summarization itself, node grouping can be applied to many graph-based tasks. CoSum [ZGGS+16] involves summarization on k-partite heterogeneous graphs to improve record linkage between data sets, otherwise known as entity resolution. CoSum transforms an input k-type graph into another k-type summary graph composed of supernodes and superedges, using links between different types to improve the accuracy of entity resolution. The algorithm jointly condenses vertices into a supernode such that each supernode consists of nodes of the same type with high similarity, and creates superedges that connect supernodes according to the original links between their constituent nodes. The resultant summary achieves better performance in entity resolution than generic approaches, especially in data sets with missing values and one-to-many or many-to-many relations.

Network clustering methods have inspired CondeNSe in many ways, as these two fields are this close to each other, there are plenty of opportunities for network clustering to contribute to the summarization domain.

CHAPTER IV

Sequences: Networks And Neural Models

Sequences are an important type of structured data that is heavily studied by the machine learning and data mining communities, as sequential data is almost everywhere: time series such as heights of ocean tides, counts of sunspots, daily price of stocks and so on; temporal sequences such as customers' shopping behaviour; and ordered data such as DNA sequences and text data. I will discuss two types of tasks in sequence modeling: sequence clustering and sequence prediction. Specifically, I will introduce my contribution to time-series and network clustering, CCTN, and a novel optimization methods for language modeling - RDLG . Readers can find the majority of Section 4.1 in [LZS+19].

4.1 CCTN: Coupled Clustering of Time-Series and Networks

Motivated by the problem of human-trafficking, where it is often observed that criminal organizations are linked and behave similarly over time, we introduce the problem of *Coupled Clustering of Time-series and* their underlying *Network*. The goal is to find tightly connected subgroups of nodes that also have similar node-specific time series (temporal—not necessarily structural—behavior). We formulate the problem as a coupled matrix factorization for the time series, combined with regularization for

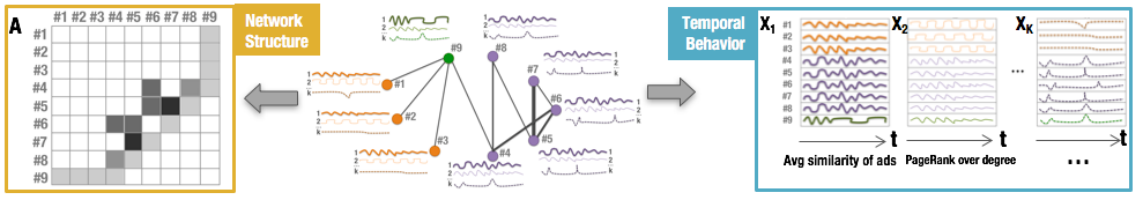


Figure 4.1: Human trafficking example: Nine phone numbers (nodes) form three clusters with tight connections, and similar temporal behaviors. The matrix on the left illustrates the representation of the network structure ($\mathbf{A}$) and the ones on the right show the different types of node-specific behaviors over time ($\mathbf{X}_k$).

network smoothness. We propose CCTN, and an incrementally-updated counterpart, CCTN-INC, which efficiently handles network updates. Extensive experiments show that CCTN is up to 4× more accurate than baselines that consider graph structure or time series alone, and CCTN-INC is up to 55× faster than CCTN. As an application, we explore an exclusive database with millions of online ads on human trafficking, and successfully deploy our technique to detect criminal organizations.

Clustering, or finding groups of similar entities, as we discussed above, is a fundamental task in data mining and machine learning. In many applications, time series and networks co-occur and may need to be clustered jointly instead of individually [BUWK15, JRSD15, DDT+16, Hes04, BGLL08].

In this work we introduce the challenging problem of **Coupled Clustering of** (entity-specific) **Time-series and their underlying Network**, where we aim to group entities into '*temporally*' and '*structurally*' coherent clusters (Fig. 4.1). Our rationale is that jointly considering network-centric and temporal (but not necessarily structural) behavioral features should lead to better clustering results than treating each data modality separately. We give two motivating examples.

Example 4.1 (Human trafficking)**.** The Internet plays a key role in both enabling and combating human-trafficking. For instance, classified ads that have contact

information for the interested parties have been shown to be useful for discovering criminal networks. Based on our analysis of human-trafficking data, we assume that (1) phone numbers belonging to the same criminal organization often co-appear in ads; and (2) ads from the same orgs have similar content (e.g., sentences, expressions). Thus, the problem of detecting criminal organizations can be framed as coupled clustering of (1) the phone number co-occurrence network *and* (2) a set of phone number-specific time series that capture ad content similarity (e.g., per-day average content similarity between the posted ads that mention the same phone number, shown as $\mathbf{X}_1$ in Fig. 4.1).

Example 4.2 (Social Networks)**.** In many cases, the topology of interactions between users is augmented with information about their temporal behaviors. For instance, in location-based social networks, the users can be characterized by their mobility or check-in patterns, whereas in scientific collaboration networks they can be described by their temporal topical interests.

In these examples, there are two constituent problems: time-series clustering and graph clustering. Both of them have been studied extensively but *separately* in the literature [BUWK15, LSSK18, JRSD15, DDT+16, Hes04, BGLL08, AGP+16, DPK+17, SSK18]. However, in these and other real scenarios, the time series correspond to entities that do *not* occur in isolation, but are *related* via an underlying network (e.g., the phone number co-occurrence network in Example 4.1). On one hand, most existing time-series clustering methods simply ignore this underlying structure or lack a principled way of incorporating it. On the other hand, most graph clustering methods aim to find tightly connected subgraphs or communities in static and dynamic networks [Sch07] by optimizing structural quantities, such as modularity or conductance [Hes04, BGLL08]. Our proposed problem of coupled

time-series and network clustering differs from dynamic graph clustering [LAL+16]: the former does not necessarily require snapshots of a graph over time; it can operate on a single network with multiple node-specific temporal behaviors or time series. It is also different from prior work that detects correlated changes in dynamic networks [CBLH12] based on the graph structure alone.

In this work we aim to fill in this gap. Specifically, we introduce the coupled clustering problem which aims to group nodes such that the similarity of their *time-series behaviors* and their *structural connectivity* is maximized per cluster. We formulate the problem as an intuitive latent time-series clustering problem joint with graph regularization, and show that it admits a standard quadratic programming solution. In more detail, our contributions are:

• **Novel Formulation:** Motivated by real applications, we propose the problem of finding groups of nodes that are both densely connected (network structure) and temporally coherent (time series of, potentially, non-structural behaviors).

• **Principled Methods:** To effectively solve the problem, we propose CCTN and its counterpart CCTN-INC that efficiently handles updates (e.g., new nodes/edges and observations in the time series).

• **Extensive Experiments:** We perform experiments on synthetic and real-world networks with up to 6.9 million edges, and show the effectiveness and efficiency of our proposed methods over the baseline methods.

• **Application:** We explore an exclusive database of millions of online ads on human-trafficking, and show the potential of CCTN in detecting criminal organizations.

The code and the supplementary material is hosted at `https://github.com/yikeliu/CCTN`.

Now we provide formal definition of our problem.

4.1.1 Proposed Problem

Let $G = (\mathcal{V}, \mathbf{E})$ be a weighted graph with $n = |\mathcal{V}|$ nodes and $m = |\mathbf{E}|$ edges, and $\mathbf{A}$ be its weighted adjacency matrix. We assume that each node i is associated with K different types of time series (e.g., K different behavioral patterns). We denote $\mathbf{X}_k \in \mathbb{R}^{n \times T}$ as the stacked matrix where row i corresponds to the $k^{\texttt{th}}$ type time-series of node i in the network.

In Example 4.1 (Fig. 4.1), the network based on human trafficking activity consists of phone numbers (nodes) that are linked if they appeared in the same online ad, with link or edge weight equal to their number of co-appearances. The first temporal behavior of each node is the per-day average content similarity between the posted ads that mention the phone number. The second behavior is its per-day $\frac{\text{PageRank}}{\text{degree}}$-ratio, which can capture unusual structural connectivity patterns.

Problem Formulation

As we mentioned above, we focus on the problem of clustering nodes that also have similar observed temporal behaviors. For consistency with the typical time-series or graph clustering problems, we make the following explicit assumptions:

(A1) **Node Temporal Behavior Similarity.** Nodes from the same cluster have similar patterns in the $k^{\texttt{th}}$ time-series type (for types $k = 1, \ldots, K$).

(A2) **Graph Smoothness.** If two nodes are connected, their cluster assignments are similar. Also, the stronger the connection between them, the more likely they are to belong to the same cluster.

These assumptions align well with our motivating example. In human-trafficking, it

Table 4.1: Major symbols and definitions.

Notation	Description
$G(\mathcal{V}, \mathbf{E})$	graph
$\mathcal{V}$, $n = \lvert\mathcal{V}\rvert$	node-set and number of nodes of G, resp.
$\mathbf{E}$, $m = \lvert\mathbf{E}\rvert$	edge-set and number of edges of G, resp.
$\mathbf{A}$	adjacency matrix of G, with entries $a(i, j)$
$\mathbf{D}$	diagonal degree matrix, $d(i, i) = \sum_j a(i, j)$ & $d(i, j) = 0$ o/w
$\mathbf{L}$	Laplacian matrix, $\mathbf{L} = \mathbf{D} - \mathbf{A}$
$\mathbf{X}_k$	$n \times T$ stacked matrix of the $q^{\texttt{th}}$ time series type per node
$\mathbf{C}$	$n \times d$ embedded-clustering matrix (with node embeddings)
$\mathbf{W}$	$d \times T$ stacked basis matrix of temporal patterns
$\mathbf{c}$	$n \times 1$ vector with the cluster assignments per node
d	dimensionality of embedding for time-series patterns
t, T	timestamp and total number of timestamps, respectively

is believed that linked phone numbers that have similar neighbors (A2) and behave similarly over time (A1)—e.g., with high average content similarity in their ads—may belong to the same criminal organization.

With these assumptions, we establish our model for the coupled clustering of time-series and network problem. We propose to embed the node-specific time-series patterns in a latent d-dimensional space using two factors: (1) a *basis* consisting of d time-series patterns which are stacked by row in matrix $\mathbf{W} \in \mathbb{R}^{d \times T}$, and (2) a matrix $\mathbf{C} \in \mathbb{R}^{n \times d}$ that describes the temporal behavior of each node as a weighted combination of the basis. We call $\mathbf{C}$ the embedded-clustering matrix, since similarities among its rows represent similar temporal behaviors among the corresponding nodes. Based on this representation, we can summarize the time-series patterns of all the nodes as follows:

$$\widetilde{\mathbf{X}}_k = \mathbf{C} \cdot \mathbf{W}, \tag{4.3}$$

where the same factors $\mathbf{W}$ and $\mathbf{C}$ are used to describe the various types of true temporal behaviors, $\mathbf{X}_k$. This model has several advantages: (1) it allows us to couple the various temporal behaviors and express them in terms of the *same basis* of temporal patterns, and (2) it constrains and guarantees a solution to our proposed problem. As shown below, if the temporal behaviors were modeled independently (i.e., via a different basis of temporal patterns $\mathbf{W}_k$ per type k), our formulation would admit *any arbitrary* solution for the embedded-clustering matrix $\mathbf{C}$, which is *not* meaningful.

By combining this model together with graph regularization, we formulate the **coupled clustering of time-series and network** problem.

Problem 4.4. Let $G(\mathcal{V}, \mathbf{E})$ be a network where each node u is associated with K different types of time-series denoted as $\mathbf{X}_k(u) \in \mathbb{R}^{1\times T}$. Then, the coupled clustering problem aims to assign to each node u a latent feature vector $\mathbf{C}(u) \in \mathbb{R}^{1\times d}$, which can then be projected to a cluster assignment $c(u) \in \mathbb{N}$, such that:

$$\arg\min_{\mathbf{C},\mathbf{W}} \{\sum_{k=1}^{K} a_k \cdot \|\mathbf{X}_k - \widetilde{\mathbf{X}}_k\|_F^2 + \lambda \cdot Tr(\mathbf{C}^T\mathbf{L}\mathbf{C})\} \tag{4.5}$$

where a_k controls the importance of the k^{th} time-series behavior, $\widetilde{\mathbf{X}}_k = \mathbf{C} \cdot \mathbf{W}$, $\mathbf{W}$ is the basis time-series matrix, $||\cdot||_F$ is the Frobenius norm of the enclosed matrix, $\mathbf{L}$ is the Laplacian matrix of G, $Tr(\cdot)$ is the trace of the corresponding matrix, and λ is a regularization parameter.

The **first term** of Eq. (4.5) represents the coupled clustering of the nodes based on their *K types of temporal behavior*. The clustering is given by our proposed model in Eq. (4.3). Our goal is to find the matrices $\mathbf{W}$ and $\mathbf{C}$ that best represent

coherent time-series clusters across all behavior types. We note that a model with different basis patterns $\mathbf{W}_k$ would lead to trivial clustering solutions: for any fixed $\mathbf{C}$, it would be possible to find a set of $\mathbf{W}_1, \ldots, \mathbf{W}_K$ that satisfy Eq. (4.5). In the human-trafficking example, the first term finds a joint latent representation for the temporal content similarity of ads ($\mathbf{X}_1$ in Fig. 4.1) and the temporal structural patterns ($\mathbf{X}_2$ or the $\frac{\text{PageRank}}{\text{degree}}$-ratio per phone number and day). The **second term** of Eq. (4.5) imposes a graph smoothness constraint over the cluster assignments of the nodes (regularization). Intuitively, it forces the temporally coherent nodes to also have strong connectivity, thus satisfying assumption (A2). The influence of each term on the final clustering depends on the **parameters** a_k, λ, which we discuss in detail in the supplemental material. In Example 4.1, this constraint 'refines' the candidate criminal organizations that have temporally coherent behaviors by attaching well-connected phone numbers to them (thus, leading to also structurally coherent clusters).

With the problem defined, I will introduce the CCTN as a solution to our problem.

Proposed Algorithm: CCTN

Optimizing Problem 4.4 requires minimization with respect to two matrices, $\mathbf{C}$ and $\mathbf{W}$. To render the problem tractable, we devise an alternating process: (1) We fix $\mathbf{C}$ and turn Problem 4.4 into a relatively easier quadratic problem; (2) We fix $\mathbf{W}$ and turn Problem 4.4 into a mixed integer programming problem (Thm 4.6).

Next, we give the equations that need to be solved for the coupled clustering of time-series and network problem. These will be the building blocks of our proposed algorithm, CCTN.

Theorem 4.6. *For a fixed clustering embedding* $\mathbf{C}$*, the basis time-series matrix* $\mathbf{W}$

is the solution to:

$$(\sum_k a_k \mathbf{C}^T\mathbf{C})\mathbf{W} = (\sum_k a_k \mathbf{C}^T\mathbf{X}_k) \tag{4.7}$$

where $\mathbf{C}^T$ *denotes the transpose of matrix* $\mathbf{C}$*.*

For a fixed basis time-series matrix $\mathbf{W}$*, the clustering embedding* $\mathbf{C}$ *can be found by solving the equation:*

$$\mathbf{C}\sum_k a_k \mathbf{W}\mathbf{W}^T + \lambda\mathbf{L}\mathbf{C} = \sum_k a_k \mathbf{X}_k\mathbf{W}^T, \tag{4.8}$$

which corresponds to a Mixed-Integer Programming problem. Equation (4.8) *is a Sylvester equation.*

Proof. See Appendix .1 in the supplemental material. □

The linear system in Eq. (4.7) can be solved by randomized Kaczmarz algorithm [SV09]. This randomized iterative method can find $\mathbf{W}$ with expected exponential rate of convergence.

To solve the Sylvester equation (4.8), we employ the scheme in [BS72] and rewrite it as an equation with Kronecker product (denoted as $\otimes$):

$$(\mathbf{I}_f \otimes \lambda\mathbf{L} + (\sum_k a_k \mathbf{W}\mathbf{W}^T)^T \otimes \mathbf{I}_n) \otimes vec(\mathbf{C}) = vec(\sum_k a_k \mathbf{X}_k\mathbf{W}^T)$$

where $vec()$ is the vectorization operator that takes a matrix and converts it to a vector by stacking its columns. The solution of $\mathbf{C}$ can be computed numerically by the Bartels-Stewart [BS72] algorithm. This algorithm first computes the Schur decomposition of the two matrices $\lambda\mathbf{L}$ and $-\sum_k a_k \mathbf{X}_k\mathbf{W}^T$ in Eq. (4.8) using a QR algorithm, and then solves the resulting triangular system via back-substitution.

Lemma 4.9. *The MIP problem of Eq.* (4.8) *has a unique solution iff the* $nf \times nf$ *matrix* $\mathbf{I}_f \otimes \lambda\mathbf{L} + (\sum_k a_k \mathbf{W}\mathbf{W}^T)^T \otimes \mathbf{I}_n$ *is invertible—i.e., if* $\mathbf{L}$ *and* $\sum_k a_k \mathbf{X}_k \mathbf{W}^T$ *do* not *have common eigenvalues.*

The computational cost of the original Bartels-Stewart [BS72] algorithm is $O(n^3)$. However, faster parallel solvers of large-scale Sylvester equations have been proposed, such as the Hessenberg-Schur method [GNVL79], and $\mathcal{H}$-matrix based sign function iteration [Bau08], where large matrices are represented by sparse hierarchical matrices. The latter method is $O(n \log^2 n)$.

Algorithm. Based on Theorem 4.6 and the transformations of its main equations described above, we propose the CCTN method, whose pseudocode is given in Algorithm 3.

Lines 5-9 describe the main part of our method, which seeks the solution in an iterative process, until convergence (line 9). In the absence of other information, the initialization of the matrices $\mathbf{C}$ and $\mathbf{W}$ is random (lines 3-4). After finding the embedded-clustering matrix $\mathbf{C}$, CCTN treats each row as an observation (which corresponds to a node) and applies a clustering technique in order to find similar nodes based on their latent representations in Eq. 4.5. In practice, any choice for clustering works for this step (e.g., k-means). We discuss our choices in the experiments.

Complexity Analysis

In each iteration of Algorithm 3, the computation is composed of two steps: update $\mathbf{W}$ (Step 1) and update $\mathbf{C}$ (Step 2). $\mathbf{W}$ is updated by randomized Kaczmarz algorithm [SV09]. The computational complexity of solving a linear system $\mathbf{M}\mathbf{x} = \mathbf{b}$ is $O(n_K t_K)$, where $\mathbf{M} \in \mathbb{R}^{m_K \times n_K}$ and t_K is the number of iterations of random Kaczmarz update. For $m_K \neq n_K$, we have $t_K = \frac{2n_K}{(1-\sqrt{y})^2} \log \frac{1}{\epsilon_K}$, where $y := \frac{n_K}{m_K}$, and

Algorithm 3 CCTN: Coupled Clust. of Time-series & Network

Input: Graph $G(\mathcal{V}, \mathbf{E})$; stacked matrices of k-type time-series $\{\mathbf{X}_k\}$ with T timesteps, parameters a_k and λ, dimensionality d
Output: Vector with cluster assignments $\mathbf{c}$

$\epsilon = 10^{-6}, \tau_{max} = 100$ // Constants for convergence
$\tau = 0$ // Iteration # initialization
$\mathbf{C}_{(\tau)} = rand(n, d)$ // $n = |\mathcal{V}|$
$\mathbf{W}_{(\tau)} = rand(d, T)$
repeat
  $\tau = \tau + 1$
  // **Step 1**: Update $\mathbf{W}$ using Eq. (4.7)
  $\mathbf{W}_{(\tau)} = (\sum_k a_k \mathbf{C}_{(\tau-\mathbf{1})}{}^T \mathbf{C}_{(\tau-1)})^{-1} (\sum_k a_k \mathbf{C}_{(\tau-1)}^T \mathbf{X}_k)$
  // **Step 2**: Update $\mathbf{C}$ by solving Eq. (4.8) following [Bau08]
  $\mathbf{C}_{(\tau)} \sum_k a_k \mathbf{W}_{(\tau-1)} \mathbf{W}_{(\tau-1)}^T + \lambda \mathbf{L} \mathbf{C}_{(\tau)} = \sum_k a_k \mathbf{X}_k \mathbf{W}_{(\tau-1)}^T$
until $(||\mathbf{C}_{(\tau)} - \mathbf{C}_{(\tau-1)}||_1 < \epsilon$ & $||\mathbf{W}_{(\tau)} - \mathbf{W}_{(\tau-1)}||_1 < \epsilon)$ or $\tau > \tau_{max}$
// **Step 3**: Assign the nodes to clusters based on the inferred embeddings in $\mathbf{C}$ (each row is an 'observation').
$\mathbf{c} =$ cluster_rows$(\mathbf{C}_{(\tau)})$
return c

ϵ_K is the accuracy of the randomized Kaczmarz algorithm. In CCTN, we have a square matrix ($\mathbf{M} = \sum_k a_k \mathbf{C}^T \mathbf{C}$), but since it has exponential convergence [SV09], t_K will be very small ($t_K \sim 5$ in practice). Updating $\mathbf{C}$ with the $\mathcal{H}$-matrix based sign function iteration [Bau08] has complexity $O(n \log^2 n)$. Hence, the complexity of CCTN is $O((dt_K + (n \log^2 n))$.

We also developed an incremental version of CCTN that handles network updates more efficiently.

4.1.2 CCTN-inc: Incremental Updates

In many applications, including our motivating application of human-trafficking, the data are changing over time: new nodes (i.e., phone numbers) and edges (new co-occurrences) are added to the network, and new timestamps are added to the behavioral time series for the existing nodes (e.g., content similarity on the new days). In our experiments, we observed that for synthetic data (Kronecker graphs) with more than 6.6k nodes, the runtime of CCTN exceeds a week. Moreover, clustering

millions of nodes is quite expensive. Thus, we propose the problem of *incremental* coupled clustering.

Problem Formulation

The problem of **Incremental Coupled Clustering** seeks to efficiently handle incremental updates in the network structure *and* the time series, so that the computation that needs to be performed per timestamp is minimized.

Problem 4.10. Let $\mathbf{A}' \in \mathbb{R}^{n' \times n'}$ and $\mathbf{X}'_k \in \mathbb{R}^{n' \times T'}$ be the augmented adjacency matrix with n' nodes and the stacked matrix of the k^{th} type time series data with T' timestamps, respectively. Let also $\mathbf{C}$ and $\mathbf{W}$ be the solutions of Problem 4.4. The Incremental Coupled Clustering problem aims to find the perturbations $\Delta\mathbf{W}$ and $\Delta\mathbf{C}$ in the matrices of basis temporal behaviors and cluster embeddings s.t. the new solutions are expressed as $\mathbf{W}' = \mathbf{W} + \Delta\mathbf{W}$ and $\mathbf{C}' = \mathbf{C} + \Delta\mathbf{C}$, respectively:

$$\arg\min_{\mathbf{C}',\mathbf{W}'} \{\sum_{k=1}^{K} a_k \cdot \|\mathbf{X}'_k - \widetilde{\mathbf{X}}'_k\|_F^2 + \lambda \cdot Tr(\mathbf{C}'^T \mathbf{L}' \mathbf{C}')\}$$

where a_k and λ remain the same as in CCTN, $\mathbf{L}'$ is the updated Laplacian matrix of $\mathbf{A}'$, and $\widetilde{\mathbf{X}}'_k = \mathbf{C}' \cdot \mathbf{W}'$.

Thus, this problem seeks to incrementally update the cluster assignment vector $\mathbf{c}'$, obtained by projecting the new embedded-clustering matrix $\mathbf{C}'$.

Incremental Algorithm: CCTN-inc

To derive the solution of the incremental problem, we leverage small perturbations $\Delta\mathbf{Z}$ for each matrix $\mathbf{Z}$ that is involved in the derivations. Specifically, we rewrite the incremental adjacency matrix as (1) the original matrix and (2) the difference-

matrix with the difference in weights between existing nodes and the connections to new nodes: $\mathbf{A}' = \mathbf{A}_{(n' \times n')} + \Delta\mathbf{A}$. In the human-trafficking example, $\Delta\mathbf{A}$ contains new phone numbers that appeared in ads published after time T, and new edges between numbers that co-appeared in ads after T.

Similarly the stacked matrix of k^{th}-type time series and the Laplacian of the new graph can be written as: $\mathbf{X}'_k = \mathbf{X}_{k,(n' \times T')} + \Delta\mathbf{X}_k$ and $\mathbf{L}' = \mathbf{D}' - \mathbf{A}' = \mathbf{L} + \Delta\mathbf{L}$. In our example, $\mathbf{X}'_k$ has additional rows for the new nodes (past the original n) and more columns for the new timestamps (past T).

Based on the above definitions, we can compute the incremental matrix $\mathbf{W}'$ by simply computing $\Delta\mathbf{W}$ and adding it to the solution of the non-incremental version. As in the solution of Problem 4.4, in the second step, we fix $\mathbf{W}'$ and find the new solution for $\mathbf{C}'$.

Theorem 4.11. *For a fixed clustering embedding $\mathbf{C}'$, the difference $\Delta\mathbf{W}$ in the stacked matrix of the d base time series patterns is given by:*

$$\Delta\mathbf{W} = \left(\sum_k a_k \mathbf{C}'^T \mathbf{C}'\right)^{-1} \sum_k a_k \mathbf{C}'^T \Delta\mathbf{X}_k \tag{4.12}$$

For a fixed basis time-series matrix $\mathbf{W}'$ (defined in Eq. (4.12)), the difference in the embedded-clustering matrix $\Delta\mathbf{C}$ is approximated by solving the following Sylvester equation:

$$\begin{aligned} \Delta\mathbf{C} \sum_k a_k \mathbf{W}\mathbf{W}^T + \lambda\mathbf{L}\Delta\mathbf{C} = \sum_k a_k (\mathbf{X}_k (\Delta\mathbf{W})^T \\ + (\Delta\mathbf{X}_k)\mathbf{W}^T - \mathbf{C}(\Delta\mathbf{W})\mathbf{W}^T - \mathbf{C}\mathbf{W}(\Delta\mathbf{W})^T + \lambda(\Delta\mathbf{L})\mathbf{C}). \end{aligned} \tag{4.13}$$

Table 4.2: Synthetic data: Description of the six cases that we designed for evaluation. For each cluster, we generate for its constituent nodes a specific type of time series per case (e.g., identical, correlated, noisy).

Case No.	Time series per cluster	Description of the clusters in terms of their nodes' time series
1	Random identical	$C_i = \{x_i(t), \ldots, x_i(t)\}$, $i \in \{1,2,3\}$, x_i randomly generated
2	Informed identical	$C_i = \{x_i(t), \ldots, x_i(t)\}$, $i \in \{1,2,3\}$, x_i extracted from real data
3	Correlated	$C_i = \{a_1(x_i(t)+b_1), \ldots, a_{\lvert C_i\rvert}x_i(t+b_{\lvert C_i\rvert})\}$, $i \in \{1,2,3\}$, x_i extracted from real data ($\lvert C_i\rvert$: size of i^{th} cluster)
4	Noisy	$C_i = \{x_i(t)+e_i(t), \ldots, x_i(t)+e_i(t)\}$, $i \in \{1,2,3\}$, x_i extracted from real data, $e_i(t) \sim \mathcal{N}(1,0)$
5	Anti-correlated	$C_i = \{x_i(t), \ldots, -x_i(t)\}$, $i \in \{1,2,3\}$, x_i extracted from real data, C_i split at half
6	Informed split	$C_{1,2} = \{x_{1,2}(t), \ldots, x_{1,2}(t)\}$, $C_3 = \{x_1(t), \ldots, x_2(t)\}$, $x_{1,2,3}$ extracted from real data, C_3 split at half

Proof. See Appendix .1 in the Appendix. □

Algorithm. Based on Theorem 4.11, we propose CCTN-INC, an effective and fast approximation of CONDENSE, which handles incremental updates in the network structure, the introduction of new nodes, and changes in the behavioral patterns of existing nodes. We give the high-level pseudocode of CCTN-INC in Algorithm 7 in the supplemental material (Appendix .2).

Complexity Analysis

Although Eq. (4.12) involves inverting a matrix, the computation is not prohibitive due to its very small size. Matrix $\mathbf{C}'$ has size $n \times d$, and thus $\mathbf{C}'^T\mathbf{C}'$ (which is the matrix that needs to be inverted) is a $d \times d$ matrix. In practice, the dimensionality d of the embedding is significantly smaller than the number of nodes n, and most likely is in the order of 10-20 features. Equation (4.13) in Theorem 4.11 is still a Sylvester equation that can be solved with the Bartels-Stewart algorithm [BS72], as in CCTN. Due to the significant sparsity of $\Delta\mathbf{C}$, the computation of the incremental matrix is sub-quadratic, $O(n \log^2 n)$ [Bau08].

4.1.3 Experiments

Our experiments are geared toward answering the following questions: **(1)** How effective is CONDENSE in terms of identifying temporally and structurally coherent

clusters? **(2)** How well does CCTN-INC approximate CCTN? **(3)** Do CCTN and CCTN-INC scale well to large datasets?

(4) Does CCTN generate intuitive clusters in real applications? **(5)** How robust is CCTN to different parameter settings? Before we present our results, we discuss our datasets, baselines and experimental setup. We answer question (5) in Appendix .5.

Data

We use both synthetic and real datasets.

Synthetic Data. We generate three cliques of different sizes (50, 100, and 200 nodes), with random, sparse connections between them. For simplicity we treat the graph as unweighted, and keep the graph structure constant over time (i.e., we use one static network). For the node-specific temporal behavior, we either randomly generate time series, or extract time series of content similarity from the real human-trafficking HT-1 data (described below), and add six types of noise (Table 4.2).

Real Data. We also use 3 exclusive real datasets: two in human-trafficking domain (HT-1/HT-1M, HT-2), and one in the military domain (MITRE).

• **Human-trafficking data 1 (HT-1, HT-1M)**: This is a **labeled** dataset of advertisements assigned to clusters. For each advertisement, we have all or part of the following information: {region, phone number, text, title, post time, age, user location, city, cluster id}. The phone numbers are assigned cluster ids by domain experts (ground truth).

We create the co-occurrence graph (temporal and aggregated) on phone numbers by adding edges between phone numbers that appear in the same ad, and weighing them by the frequency of their co-occurrence. HT-1M denotes the manipulated

Table 4.3: Real data

Dataset	Nodes	Edges	Timestamps	Description
HT-1	60 437	1 241 773	13	labeled, human-trafficking
HT-2	61 155	129 196	60	unlabeled, human-trafficking
MITRE	3 813	6 892 425	335	labeled, Twitter data

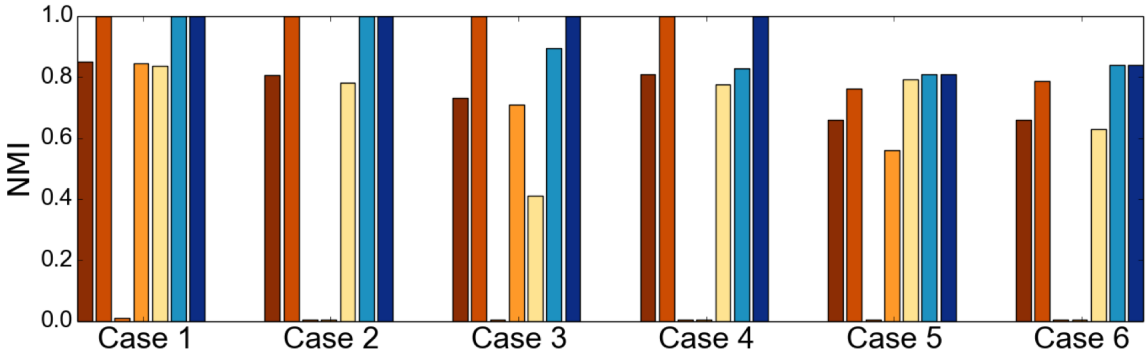
graph of HT-1 where attackers randomly connect their phone numbers to public phone numbers such as AT&T service number.

The node-specific temporal behaviors consist of: (1) Structural time series of $\frac{\text{PageRank}}{\text{degree}}$-ratio, which can capture anomalous patterns and is obtained from the temporal graphs—i.e., $\mathbf{X}_1 = \mathbf{X}_{struc}$; (2) Content time series for the average pairwise Jaccard index across the ads with the same phone number (during the same time interval)—i.e., $\mathbf{X}_2 = \mathbf{X}_{cont}$. Although we use $k = 2$ temporal behaviors, CCTN can scale with greater k.

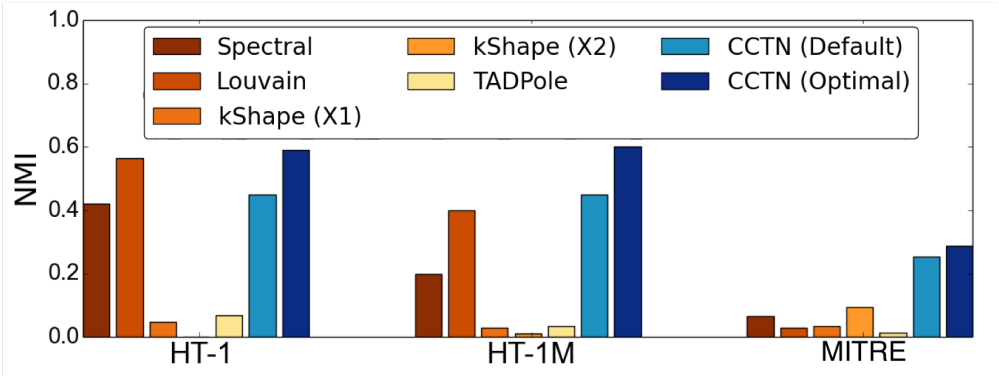
• **Human-trafficking data 2 (HT-2)**: This is an **unlabeled** dataset of ads, for which we have all or part of the following information: {description, uri, date of creation, contact information, name, location}. Graphs and time series are generated the same way as for HT-1.

• **MITRE data**: This **labeled** Twitter dataset [ZSY+15] is annotated with both GEO-location and social event forecasting results. Each location is assigned to a cluster by domain experts, based on event types and users. We generate a graph that consists of locations (nodes) connected by edges that are weighted by their geographic distance (based on longitude and latitude). Edges with distances above $d = 1000$ miles are pruned.

The temporal behaviors include: (1) the structural time-series of the $\frac{\text{PageRank}}{\text{degree}}$-ratio—i.e., $\mathbf{X}_1 = \mathbf{X}_{struc}$; and (2) the activity-specific behavior consisting of the count of



(a) Synthetic data: CCTN performs better than the baselines on difficult cases or equally well on easy cases.



(b) Real data: Overall, CCTN performs consistently better than or equally well as the baseline methods.

Figure 4.2: Accuracy of CCTN and CCTN-INC on synthetic and real data.

events at each location over time—i.e., $\mathbf{X}_2 = \mathbf{X}_{event}$.

Baselines

We choose methods that fall into the two subproblems that we solve: (1) time-series clustering, from which we pick two recent, best-performing works, k-Shape [PG15] and TADPole [BUWK15]; and (2) graph clustering on the aggregated graph, from which we choose spectral clustering [Hes04] and Louvain [BGLL08]. We describe the baselines in Appendix .3.

Experimental Setup

For CCTN and CCTN-INC, practitioners can choose any clustering method that takes in feature vectors of the CCTN node embeddings in $\mathbf{C}$ (line 10 of Alg. 3). For simplicity, we exploit the widely-used k-means clustering [Llo82] for labeled data,

and x-means [Jai10] for unlabeled data with unknown number of clusters. To show the effect of parameters on the performance of CCTN, we report the results on two variants: (1) the '*default*' case, where we set $a_1 = a_2 = 1$, $\lambda = 0.01$, $d = 3$; and (2) the '*best*' case, where the parameters are chosen via grid search on a small random subset of the data. The sample we used in our experiments consists of $1/10^{th}$ of the data for MITRE, and $1/100^{th}$ of the HT-1 and HT-1M data. We performed grid search for the following ranges: $a_1 \in (0, 10], a_2 \in (0, 10], \lambda \in (0, 10], d \in (0, 10]$ with an interval of two, hence the 'best' is not the globally best result. Results on CCTN's robustness to parameter settings are in Appendix .5. We ran the CondeNSe until convergence (line 9 in Alg. 3) for all the datasets except for HT-1, for which we set $\tau_{max} = 10$ iterations.

For the baselines we used the default values of their parameters: resolution $\tau_L = 10^{-4}$ for Louvain, band size 0.08 for DTW, and cut off distance 1.4619 for TADPole.

All the experiments were run on Intel(R) Xeon(R) CPU E5-1620 v3 @ 3.50GHz with 264GB RAM.

Evaluation Since our objective is to find coherent groups of nodes, we focus on the evaluation of the embedded-clustering matrix $\mathbf{C}$ and the clustering results. We omit a detailed analysis of the auxiliary variable $\mathbf{W}$ due to space limitations. For the performance of cluster recovery, we use (1) normalized mutual information (NMI) and (2) rand index, which measure the agreement between the found and ground-truth clusters, but capture different information. The results are generally consistent across the two metrics; for brevity we give the results based on rand index in Appendix .4.

Accuracy

(1) CCTN. First, we investigate how CCTN compares to the baselines in terms of

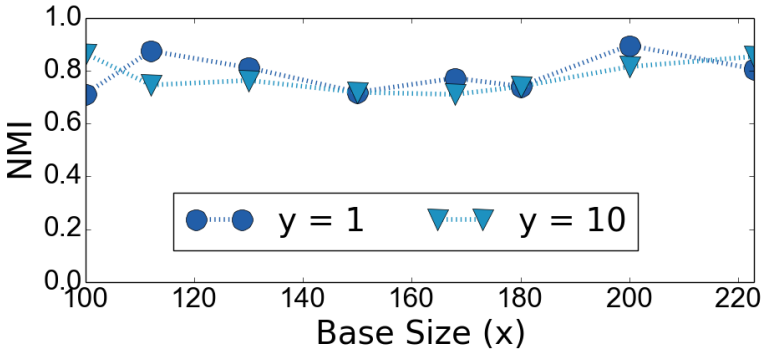
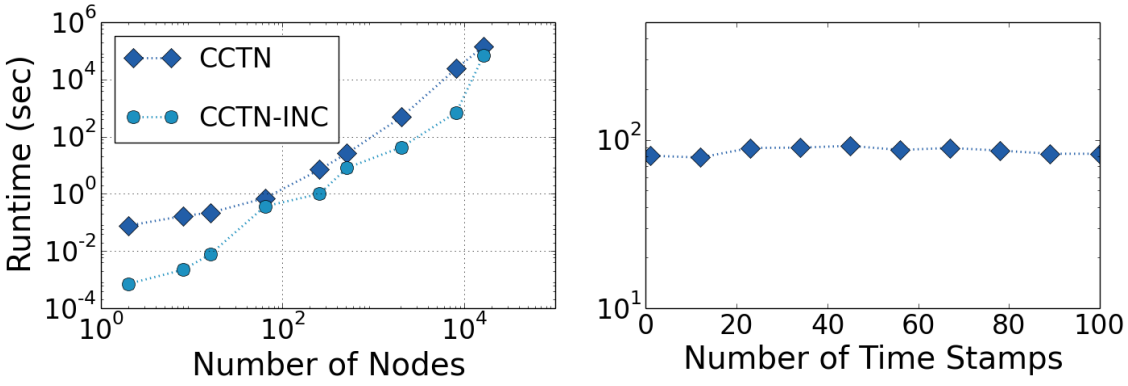


Figure 4.3: MITRE: CCTN-INC approximates CCTN well. They yield similar clusterings.

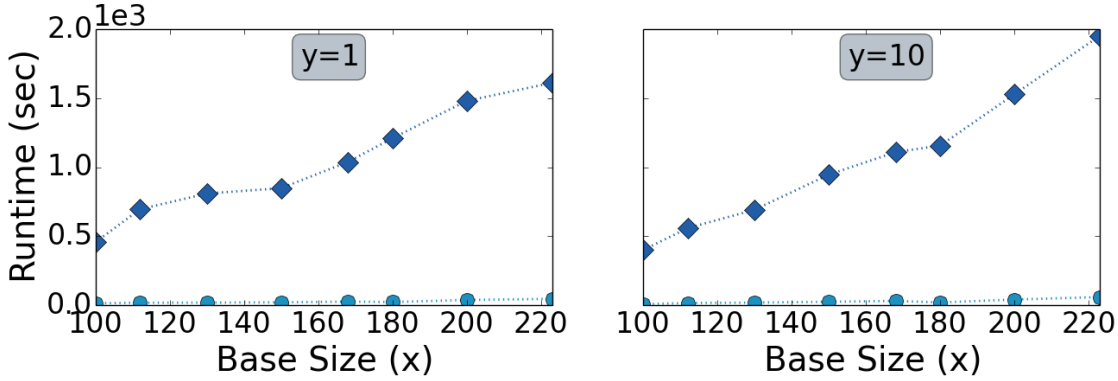
effectiveness in identifying temporally and structurally coherent clusters. We present the NMI of the clusterings that all methods produce on the synthetic datasets in Fig. 4.2a and on the real datasets in Fig. 4.2b. We report the results based on the rand index measure in Fig. A.1 (Appendix .4). For the synthetic data, where the structural time series are constant (due to the static network), TADPole is less effective when leveraging $\mathbf{X}_{struc}$ in addition to $\mathbf{X}_{cont}$. For that reason, we only report its results using $\mathbf{X}_{cont}$.

Observation 1. CCTN outperforms the baselines on the real data, and is sometimes tied with Louvain on the synthetic data. The baselines have more variable performance across datasets.

Louvain, which usually outperforms spectral clustering, gives equally good results for HT-1, but has significantly inferior performance on HT-1M and MITRE. Since it only relies on the graph structure, it is heavily impacted by graph noise, as shown by its performance on the manipulated HT-1M dataset (Fig. 4.2b). K-Shape and TADPole, which ignore the graph structure, tend to perform worse than CCTN, especially on the real datasets ($NMI < 0.1$). By combining the temporal and structural aspects, CCTN is more robust: it achieves up to $NMI = 0.6$ for real data, and often perfect performance in the synthetic cases.



(a) Runtime vs. number of nodes / number of time stamps



(b) MITRE: Runtime of CCTN-INC vs. CCTN.

Figure 4.4: Runtime analysis.

(2) CCTN-inc. Second, we explore how well CCTN-INC approximates the clusterings that our exact approach, CCTN, generates. To that end, we split the real labeled data into two parts: (1) the first x timestamps (or the 'base size'), which are used to create the input graph and time series, as described above; and (2) the future timestamps. Then, we compute the agreement between the clustering that CCTN-INC outputs after incremental updates for y timestamps, and the clustering of CCTN when applied to the $(x + y)$ timestamps all together.

For brevity, in Fig. 4.3, we show the agreement (NMI) of the two methods on MITRE by varying the base size x, and for $y \in \{1, 10\}$ steps of incremental updates. The results based on rand index are given in Fig. A.2 (Appendix .4). The results are

consistent for other values of y and the other datasets.

Observation 2. CCTN-INC is a good approximation of CCTN (wrt both NMI and rand index), independent of the incremental interval y. The accuracy varies for different base size x, but remains relatively stable. As expected, performing more incremental updates (e.g., $y = 10$) leads to lower, but still high, agreement.

We note that in this experiment we do not compare the CCTN-INC clustering with the ground-truth labels, because its performance is measured by how well it approximates CCTN and how efficient it is (see above).

Runtime

(1) CCTN. Third, we evaluate how well CCTN scales with the size of the data, and specifically with (i) the number of nodes in the aggregated graph, and (ii) the number of timestamps.

For experiment (i), we fix the number of timestamps to 10, and generate the aggregated graph by combining 10 Kronecker graphs [LCK+10] of different sizes (given random seed matrices). Assuming $\sqrt{n/2}$ clusters (which is the rule of thumb for choosing number of clusters [cyt]), we generate that many random time series, and randomly assign them to the n nodes.

For experiment (ii), we generate synthetic data in the same way, but fix the number of nodes to 1 024 and vary the number of timestamps of the time series. We demonstrate the results in Fig. 4.4a.

Observation 3. The runtime of CCTN increases subquadratically with the number of nodes, and smooths out to near-linear when the number of nodes becomes large. Its efficiency is independent of the number of timestamps T (right plot in Fig. 4.4a).

(2) CCTN-inc. Finally, we compare CCTN-INC to CCTN to show its efficiency

benefits. Figure 4.4b shows the runtime of the two methods on MITRE for different combinations of x and y (explained above).

Observation 4. CCTN-INC is up to $30-55\times$ faster than CCTN, due to the sparser matrix computations that it performs per update, and its faster convergence.

CCTN-INC usually converges in $1/50 \sim 1/30$ of the iterations that CCTN needs for real data. This contributes to the significant reduction in runtime.

4.1.4 Case Study on Real Data

To evaluate whether CCTN finds intuitive clusters in real applications, we apply it to the human-trafficking HT-2 dataset. Among the 61 155 nodes in HT-2, CCTN identified 15 clusters, with size ranging from 129 to 31 060 nodes. All the clusters have numerous phone number pairs that have appeared in similar ads.

In Fig. 4.5, we show a randomly-picked example consisting of phone numbers 1-3236****** and 1-3232******. Across the 60 timestamps, these phone numbers have co-occurred 2 656 times in different ads (high edge weight in the graph). Besides the high similarity between the ads they co-occurred in, we also observe that many of the ads in which each phone number appeared alone have largely similar content (both in text and images). In the bottom ad, which was posted on Sep 1, 'GRAND OPEN' suggests that the service was just opened. The top ad was posted several months later, on Dec 27, yet it has very similar text and pictures to the bottom ad (similar text is marked with the same color). Though the original data extraction process did not identify the ads to be at the same location (Long Beach vs. the general Los Angeles area), CCTN has identified two phone numbers pointing to exactly the same address in the ads, which is clear evidence that they belong to the same organization. Similar observations for many other pairs of phone numbers hold.

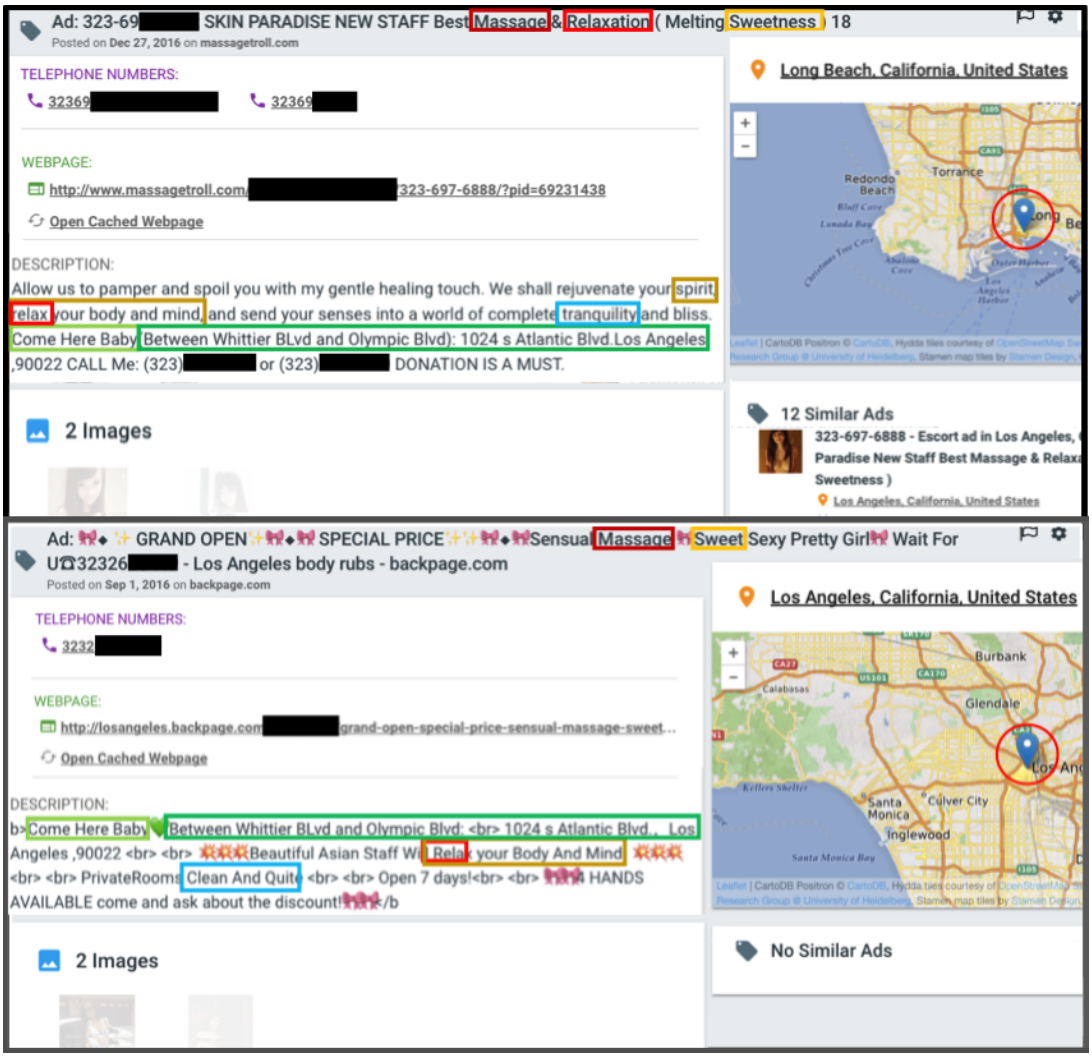


Figure 4.5: Advertisements related to the phone numbers 1-3236****** and 1-3232******.

In a nutshell, our proposed approach has three main advantages compared to the state-of-the-art approaches: (1) it combines the strengths of both model-based similarity measurement and low-dimension representation learning (i.e., cluster embedding); (2) it is built upon a unified framework that performs representation learning, coherence evaluation, and clustering simultaneously; and (3) it utilizes rich metadata information (e.g., co-occurrence graphs) to improve the quality of clustering.

To conclude CCTN: motivated by the need to identify criminal organizations involved in human-trafficking (which are often related to each other, and behave sim-

ilarly over time), we introduced the problem of *coupled clustering of time-series and their underlying network*. We formulated it as an optimization over the time-series embeddings, coupled with graph regularization. To solve it, we proposed CCTN, an efficient method that combines matrix factorization and network embeddings, as well as an incrementally-updated counterpart that efficiently adjusts the discovered clusters to the graph and temporal changes over time. Our experiments on synthetic and large real data showed that our methods are up to 4× more accurate than the baselines that ignore either the graph structure or the time-series component. We also demonstrated that CCTN produces sensible results on real human-trafficking data and identifies temporally and structurally coherent clusters, which likely represent criminal organizations.

4.2 Recurrent Deep Learning Games: Optimizing RNN Training

Apart from sequence clustering we discussed in the previous section, there are numerous other tasks in modeling sequences. In this section, we focus on another major task in sequence modeling - sequence prediction, it is: given a sequence and predict the next element; or given an observation sequence and predict a hidden sequence. In this section I focus on the former task. Sequence prediction is widely applied in industry: in traffic domain it can be used to estimate travel time [WFY18]; in natural language processing (NLP) domain this task is usually referred to as language modeling and it can be applied on different levels [DMBM15].

Recurrent Neural Networks, or RNNs have dominated the field of sequence modeling for years, due to their flexibility to accommodate changing sequence lengths, and the ability of automatically learning the temporal correlations [MKB+10, GMH13, SP97, PNI+18]. Only recently there has appeared more work on attention mech-

anism and multiple NLP models have shown out-performance of transformers over RNNs [DCLT18, RNSS18]. In this thesis I still focus on RNNs with its training strategies and demonstrate my study on its optimization. In this section, networks refer to neural networks, which is different from Chapter III, where networks refer to data structures composed of nodes and edges. We also note that notations in this section is independent from the rest of the thesis. My contributions are as follows:

- Design a one-shot simultaneous move game RDLG with infinite action sets for a corresponding optimization problem RDLP on recurrent neural network architecture.
- For RDLG and RDLP , we give theoretical proof of equivalence between a Nash equilibrium and a global minimum in convex setting. We also provide algorithm for solving a Nash equilibrium.
- We show convergence of regret matching with experiments on language modeling, as well as comparison with baseline algorithms for solving for RDLP , and provide our analysis.

Recurrent Neural Network

Taking the definition from [GBC16], RNNs are a family of neural network for processing sequential data, i.e. a sequence of values $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\tau)}$. They can scale to much longer sequences than would be practical for networks without sequence-based specialization. Most recurrent networks can also process sequences of variable length. RNNs are often dealt with unrolling with the key idea of parameter sharing. A wide variety of recurrent neural networks are designed for different tasks. Some examples of important design patterns for recurrent neural networks, in this thesis we focus on the following one:

- Recurrent networks that produce an output at each time step and have recurrent connections between hidden units, illustrated in Figure 4.6.

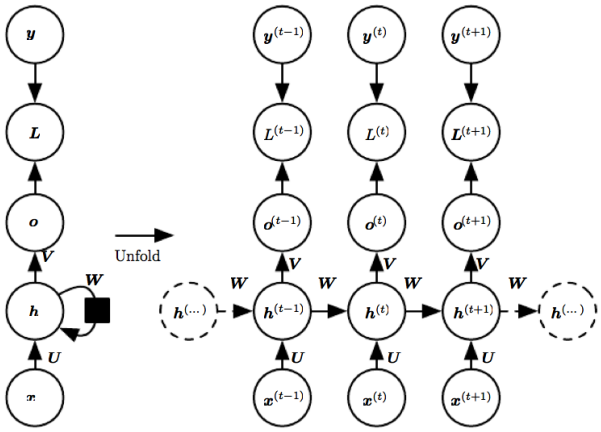


Figure 4.6: The computational graph to compute the training loss of a recurrent network that maps an input sequence of **x** values to a corresponding sequence of output **o** values. Figure from [GBC16]

Training RNNs is notoriously expensive, as it basically applies the back-propagation [GBC16] on the rolled out network (right of Fig. 4.6), depending on the length of entire sequence τ, this is called back-propagation through time (BPTT). Once rolled out, any gradient-based algorithms can be used to train an RNN. However, when the sequence length τ is very long, the computation to take the backward pass and update the parameters with gradients increases exponentially.

It is also well known that gradients propagated over very deep networks tend to vanish (mostly) or explode. This is caused by the multiplication of many Jacobians that are either very small or large in terms of value. ResNet [HZRS16] achieved huge success on the solution of gradient vanishing by introducing residual layers. Same problems exist for the rollout network of RNNs where τ is very large, where the long-term dependencies is very hard to learn.

Game Theory

Game theory is the study of mathematical models of strategic interaction between

rational decision-makers, which is widely researched by the societies of mathematics, computer science, and economics. In the classical non-cooperative setting, the Nash equilibrium (NE) refers to a proposed solution of a game involving two or more players in which each player is assumed to know the equilibrium strategies of the other players, and no player has anything to gain by changing only their own strategy [DRS09]. Compared to other fields, researchers in machine learning focus more on numerical methods for solving for NE [B$^+$14, JQV07, BS17]. Resolutions are usually graphical-based and computed via linear/linear complementarity programming. For large-scaled games, recent contributions are made on the refinement on NE finding, such as game abstraction and pruning strategies, all following the principle of regret minimization [ZJBP08, BM07].

With the proposed problem and algorithm, I intend to resolve the difficulties of RNN training on long sequences with a game-theory-based approach. In the following I state the formal definition of the RDLP problem as well as its solution - regret matching.

4.2.1 RDLP : Recurrent Deep Learning Problem

Definition 4.14 (RDLP : Recurrent Deep Learning Problem)**.** Consider the example in Fig. 4.6, for each time step from $t = 1$ to $t = \tau$, apply the following update equations, we ignore the biases for now. Consider sequential input vectors $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\tau)}\}$, the network performs computations on them with matrix parameters $\mathbf{U}, \mathbf{V}, \mathbf{W}$:

$$\mathbf{a}^{(t)} = \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} \tag{4.15}$$

$$\mathbf{h}^{(t)} = f_v(\mathbf{a}^{(t)}) \tag{4.16}$$

$$\mathbf{o}^{(t)} = \mathbf{V}\mathbf{h}^{(t)} \tag{4.17}$$

$$\hat{\mathbf{y}}^{(t)} = g_v(\mathbf{o}^{(t)}) \tag{4.18}$$

The total loss for a given sequence of $\mathbf{x}$ values paired with a sequence of $\mathbf{y}$ values would then be just the sum of the losses over all the time steps. Given a loss function $l(\mathbf{z}, \mathbf{y})$ that is convex in the first argument satisfying $0 < l(\mathbf{z}, \mathbf{y}) < \inf$ for all $\mathbf{z} \in \mathbb{R}^n$, define $l^{(t)}(\mathbf{z}) = l(\mathbf{z}, \mathbf{y}^{(t)})$ and $L^{(t)}(\mathbf{U}, \mathbf{V}, \mathbf{W}) = l^{(t)}(\hat{\mathbf{y}}^{(t)})$. The training problem is to find a set of $\{\mathbf{U}, \mathbf{V}, \mathbf{W}\}$ that minimizes $L(\mathbf{U}, \mathbf{V}, \mathbf{W}) = \tau^{-1} \sum_{t=1}^{\tau} L^{(t)}$. l is allowed to take various forms, e.g. negative log-likelihood of $\mathbf{y}^{(t)}$ given $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\tau)}$ that is convex. f_v and g_v are activation functions, for the example in Figure 4.6 we can choose $f_v = \tanh$ and $g_v = \text{softmax}$.

Similarly to the feedforward networks [SZ16], given a training input $\mathbf{x}^{(t)} \in \mathbb{R}^m$, the computation of the recurrent network is expressed by a circuit value function $c^{(t)}$ that assigns values to each vertex based on the partial order over vertices:

for $v \in x$, $c^{(t)}(v, \mathbf{U}, \mathbf{V}, \mathbf{W}) = x_v$;

for $v \in h$,

$$c^{(t)}(v, \mathbf{U}, \mathbf{V}, \mathbf{W}) = f_v\Big(\sum_{u:(u,v)\in E} c^{(t-1)}(u, \mathbf{U}, \mathbf{V}, \mathbf{W}) W(u, v) + \sum_{u':(u',v)\in E} c^{(t)}(u', \mathbf{U}, \mathbf{V}, \mathbf{W}) U(u', v) \Big)$$

,

for $v \in o$, $c^{(t)}(v, \mathbf{U}, \mathbf{V}, \mathbf{W}) = g_v(\sum_{u:(u,v)\in E} c^{(t)}(u, \mathbf{U}, \mathbf{V}, \mathbf{W})V(u, v))$.

Let $\mathbf{c}^{(t)}(o, \mathbf{U}, \mathbf{V}, \mathbf{W})$ denote the vector of the values at the output vertices, i.e. $(\mathbf{c}^{(t)}(o, \mathbf{U}, \mathbf{V}, \mathbf{W}))_k = \mathbf{c}^{(t)}(o_k, \mathbf{U}, \mathbf{V}, \mathbf{W})$.

For RDLP , we define a corresponding game RDLG .

Definition 4.19 (RDLG : Recurrent Deep Learning Game)**.** We define a one-shot simultaneous move game with infinite action sets; we specify the players, action sets, and utility functions as follows.

Players: The players consist of a protagonist p for each $v \in h \cup o$, an antagonist a, and a set of self-interested zannis s_v, one for each vertex $v \in x \cup h \cup o$.

Actions: The protagonist for vertex v chooses a parameter function U_v, W_v, V_v. The antagonist chooses a set of τ vectors and scalars $\{\mathbf{a}^{(t)}, b^{(t)}\}_{t=1}^{\tau}$, $\mathbf{a}^{(t)} \in \mathbb{R}^n$, $b^{(t)} \in \mathbb{R}$, such that $\mathbf{a}^{(t)\intercal}\mathbf{z} + b^{(t)} \leq l^{(t)}(\mathbf{z})$ for all $\mathbf{z} \in \mathbb{R}^n$; that is, the antagonist chooses an affine minorant of the local loss for each example in the training sequence. Each zannni s_v chooses a set of 2τ scalars $\{q_{vt}, d_{vt}\}$, $q_{vt} \in \mathbb{R}, d_{vt} \in \mathbb{R}$, such that $q_{vt}z + d_{vt} \leq f_v(z)$ for all $v \in x \cup h, z \in \mathbb{R}$ and $q_{vt}z + d_{vt} \leq g_v(z)$ for all $v \in o, z \in \mathbb{R}$. That is, the zanni chooses an affine minorant of its local activation function f_v or g_v for each example in the training sequence. All players make their action choice without knowledge of the other player's choice.

Utilities: For a joint action $\sigma = (\mathbf{U}, \mathbf{V}, \mathbf{W}, \{\mathbf{a}^{(t)}, b^{(t)}\}, \{q_v^{(t)}, d_v^{(t)}\})$, the zannis' utilities are defined recursively following the partial order on vertices. First, for each $i \in x$ the utility for zanni s_i on training example t is $U_i^{s(t)}(\sigma) = d_i^{(t)} + q_i^{(t)}x_i^{(t)}$; for each $v \in h$ the utility for zanni s_v on example t is $U_v^{s(t)}(\sigma) = d_v^{(t)} + q_v^{(t)}(\sum_{u:(u,v)\in E} U_u^{s(t-1)}(\sigma)W(u, v) + \sum_{u':(u',v)\in E} U_{u'}^{s(t)}(\sigma)U(u', v))$; and for each $v' \in o$ the utility for zanni s_v on example t

is $U_{v'}^{s(t)}(\sigma) = d_{v'}^{(t)} + q_{v'}^{(t)} \sum_{u:(u,v')\in E} U_u^{s(t)}(\sigma) V(u, v')$. The total utility for each zanni s_v is given by $U_v^s(\sigma) = \sum_{t=1}^{\tau} U_v^{s(t)}(\sigma)$ for $v \in x \cup h \cup o$. The utility for the antagonist a is then given by $U^a(\sigma) = \tau^{-1} \sum_{t=1}^{\tau} U^{a(t)}(\sigma)$ where $U^{a(t)}(\sigma) = b_t + \sum_{k=1}^{n} a_k^{(t)} U_{o_k}^{s(t)}(\sigma)$. They utility for all protagonists are the same, $U^p(\sigma) = -U^a(\sigma)$.

For the unconstrained problem defined above, we give the following lemma and theorem. Readers may refer to Appendix .6 for proofs.

Lemma 4.20. *Given a fixed protagonist action* $\{\mathbf{U}, \mathbf{V}, \mathbf{W}\}$*, there exists a unique joint action for all agents* $\sigma = (\mathbf{U}, \mathbf{V}, \mathbf{W}, \{\mathbf{a}^{(t)}, b^{(t)}\}, \{q_v^{(t)}, d_v^{(t)}\})$ *where the zannis and the antagonist are playing best response to p. Moreover,* $U^p(\sigma) = -L(\mathbf{U}, \mathbf{V}, \mathbf{W})$*,* $\nabla_{\mathbf{M}} U^p(\sigma) = -\nabla_{\mathbf{M}} L(\mathbf{U}, \mathbf{V}, \mathbf{W})$*, where* $\mathbf{M} \in \mathbf{U}, \mathbf{V}, \mathbf{W}$*. Given some protagonist at* $v \in h \cup o$*, if we hold all other agents' strategies fixed,* $U^p(\sigma)$ *is an affine function of the strategy of the protagonist at* v*. We define* σ *as the joint action expansion for* $\{\mathbf{U}, \mathbf{V}, \mathbf{W}\}$*.*

Theorem 4.21. *The joint action* $\sigma = (\mathbf{U}, \mathbf{V}, \mathbf{W}, \{\mathbf{a}^{(t)}, b^{(t)}\}, \{q_v^{(t)}, d_v^{(t)}\})$ *is a Nash equilibrium (NE) of the RDLG iff it is the joint action expansion for* $\{\mathbf{U}, \mathbf{V}, \mathbf{W}\}$ *and* $\{\mathbf{U}, \mathbf{V}, \mathbf{W}\}$ *is a global minimum of the RDLP .*

In practice we always bound parameters $(\mathbf{U}, \mathbf{V}, \mathbf{W})$ within some constant range for the convenience of computation. In fact, Theorem 4.21 is still valid (and most statements in Lemma 4.20) in bounded case, and the correspondence between an NE and a global minimum would be sufficient for us to apply the algorithm below.

Learning Algorithms

Similarly to [SZ16], we find the global minimum by training independent protagonist agents at each vertex against a best response antagonist and best response zannis.

Algorithm Outline. On round t, the t^{th} training example is chosen. For each

$v \in h \cup o$, each protagonist v selects her actions (U_v, V_v, W_v) deterministically. The antagonist and zannis then select their actions, which are best responses to (U_v, V_v, W_v) and to each other. The protagonist utilities U_v^p are then calculated. Given the zanni and antagonist choices, U_v^p is affine in the protagonist's action, and also by Lemma 4.20 for all $e \in E_v$, we have $\frac{\partial L^t}{\partial \omega_e} = -\frac{\partial U_v^p(U_v, V_v, W_v)}{\partial \omega_e}$. Each protagonist $v \in h \cup o$ then observes their utility and uses this to update their strategy. See Algorithm 4 for specific updates of regret matching. $\mathbf{H}$ is the convex hull basis of constraints on matrices $\mathbf{U}, \mathbf{V}, \mathbf{W}$. For convenience, we consider the bounded case as we can always scale up to proper magnitude, including the unconstrained case, i.e. the entire vector space. In this case the parameters are constrained to a cube, and the convex hull matrices are constant matrices:

$$H = \begin{bmatrix} 1 & -1 & \dots & -1 \\ 1 & 1 & \dots & -1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & -1 \end{bmatrix}$$

Like backpropagation for feedforward networks, we compare regret matching with gradient-based methods for RNNs, i.e. backpropagation through time (BPTT) on vanilla recurrent neural networks as in Algorithm 5. It was observed that the back-propagation dynamics caused the gradients in an RNN to either vanish or explode. Long Short-Term Memory (LSTM) models, as another baseline, were designed to mitigate the vanishing gradient problem, the architecture is described in Figure 4.7. In [SZ16] Exponential Weighted Average (EWA) was also used for finding the NE as well, but it did not outperform plain back-propagation.

For RM, the parameters are updated with the updated $\rho_v^{(k+1)}$ as follows:

$$\mathbf{U}^{(k+1)} = \mathbf{H}\rho_U^{(k+1)} \tag{4.22}$$

Algorithm 4 Regret Matching (RM)

Unfold the network to contain τ instances of $(\mathbf{U}, \mathbf{V}, \mathbf{W})$
$k = 0, \mathbf{h}^{(0)} = \mathbf{0}$
repeat
    $k \leftarrow k + 1$
    Observe $\mathbf{h}^{(0)}$ and $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\tau)}$
    Antagonist and zannis choose best responses which ensures $\nabla U_v^p(\mathbf{U}, \mathbf{V}, \mathbf{W}) = -\nabla L(\mathbf{U}^{(k)}, \mathbf{V}^{(k)}, \mathbf{W}^{(k)})$
    $g_v^{(k)} \leftarrow \nabla U_v^p(\mathbf{U}, \mathbf{V}, \mathbf{W})$
    For all $v \in x \cup h \cup o$ apply update $r_v^{(k+1)} \leftarrow r_v^{(k)} + \mathbf{H}^\top g_v^{(k)} - \rho_v^{(k)\top} \mathbf{H}^\top g_v^{(k)}$ $\rho_v^{(k+1)} \leftarrow (r_v^{(k+1)})_+ / (\mathbf{1}^\top (r_v^{(k+1)})_+)$
    Update all the weights in $(\mathbf{U}, \mathbf{V}, \mathbf{W})$
until stopping criteria is met

Algorithm 5 BackPropagation Through Time (BPTT)

Unfold the network to contain τ instances of $(\mathbf{U}, \mathbf{V}, \mathbf{W})$
$\mathbf{h}^{(0)} = \mathbf{0}$
Set the network inputs to $\mathbf{h}^{(0)}, \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\tau)}$
repeat
    $\hat{\mathbf{y}}^{(1)}, \ldots, \hat{\mathbf{y}}^{(\tau)}$ = forward-propagate the inputs over the unfolded network
    Back-propagate the losses $l^{(1)}(\hat{\mathbf{y}}^{(1)}), \ldots, l^{(\tau)}(\hat{\mathbf{y}}^{(\tau)})$ back across the unfolded network
    Sum the weight changes in the τ instances of $(\mathbf{U}, \mathbf{V}, \mathbf{W})$ together
    Update all the weights in $(\mathbf{U}, \mathbf{V}, \mathbf{W})$
until stopping criteria is met

$$\mathbf{V}^{(K+1)} = \mathbf{H} \rho_V^{(k+1)} \tag{4.23}$$

$$\mathbf{W}^{(k+1)} = \mathbf{H} \rho_W^{(k+1)} \tag{4.24}$$

We give the detailed weight updates of BPTT as follows:

$$\mathbf{U}^{(k+1)} = \mathbf{U}^{(k)} - \eta \nabla_{\mathbf{U}} L = \mathbf{U}^{(t-1)} - \eta \sum_{t=1}^{\tau} \mathrm{diag}(1 - (\mathbf{h}^{(t)})^2)(\nabla_{\mathbf{h}^{(t)}} L)\mathbf{x}^{(t)\top} \tag{4.25}$$

$$\mathbf{V}^{(K+1)} = \mathbf{V}^{(k)} - \eta \nabla_{\mathbf{V}} L = \mathbf{V}^{(t-1)} - \eta \sum_{t=1}^{\tau} (\nabla_{\mathbf{o}^{(t)}} L)\mathbf{h}^{(t)\top} \tag{4.26}$$

$$\mathbf{W}^{(k+1)} = \mathbf{W}^{(k)} - \eta \nabla_{\mathbf{W}} L = \mathbf{W}^{(t-1)} - \eta \sum_{t=1}^{\tau} \text{diag}(1 - (\mathbf{h}^{(t)})^2)(\nabla_{\mathbf{h}^{(t)}} L)\mathbf{h}^{(t-1)\top} \tag{4.27}$$

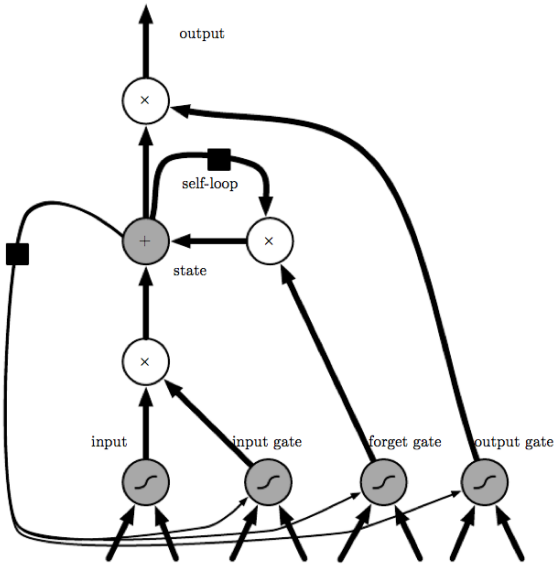


Figure 4.7: Block diagram of the LSTM recurrent network cells.Figure from [GBC16]

In LSTMs, cells are connected recurrently to each other, replacing the usual hidden units of ordinary RNNs. An input feature is computed with a regular artificial neuron unit. Its value can be accumulated into the state if the sigmoid input gate allows it. The state unit has a linear self-loop whose weight is controlled by the forget gate. The output of the cell can be shut off by the output gate. All the gating units have a sigmoid nonlinearity, while the input unit can have any squashing nonlinearity. The state unit can be used as an extra input to the gating units. The black square indicates a delay of a single time step. The self-loop weight is controlled by a forget gate unit $f_i^{(t)}$, that sets this weight to a value between 0 and 1 via a sigmoid unit:

$$f_i^{(t)} = \sigma(\sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}) \tag{4.28}$$

The LSTM cell internal state is thus updated as follows, but with a conditional

self-loop weight $f_i^{(t)}$:

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma(\sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}) \tag{4.29}$$

The external input gate unit $g_i^{(t)}$ is computed similarly to the forget gate but with its own parameters:

$$g_i^{(t)} = \sigma(\sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}) \tag{4.30}$$

The output $h_i^{(t)}$ of the LSTM cell can also be shut off, via the output gate $q_i^{(t)}$, which also uses a sigmoid unit for gating:

$$h_i^{(t)} = f_v(s_i^{(t)}) q_i^{(t)} \tag{4.31}$$

$$q_i^{(t)} = \sigma(\sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}) \tag{4.32}$$

Output of each sample is given by:

$$\hat{y}_i^{(t)} = g_v(o_i^{(t)}) = g_v(\sum_j V_{i,j}^o h_j^{(t)}) \tag{4.33}$$

Training of LSTMs still follows the rule of BPTT, only that the components of $\ldots, \mathbf{h}^{(t-1)}, \mathbf{h}^{(t)}, \ldots$ and $\ldots, \mathbf{x}^{(t-1)}, \mathbf{x}^{(t)}, \ldots$ are replaced by the LSTM block and the back propagation follows the update rules of it, where we learn the parameters $\{\mathbf{U}, \mathbf{V}, \mathbf{W}\}$ and three sets of $\{\mathbf{U}, \mathbf{W}\}$ for each gate. Gated recurrent units (GRUs, see [CVMG+14]) is a similar gate scheme for RNN architectures, we limit the baselines to LSTM as its more popular and usually has better performance on language modeling tasks.

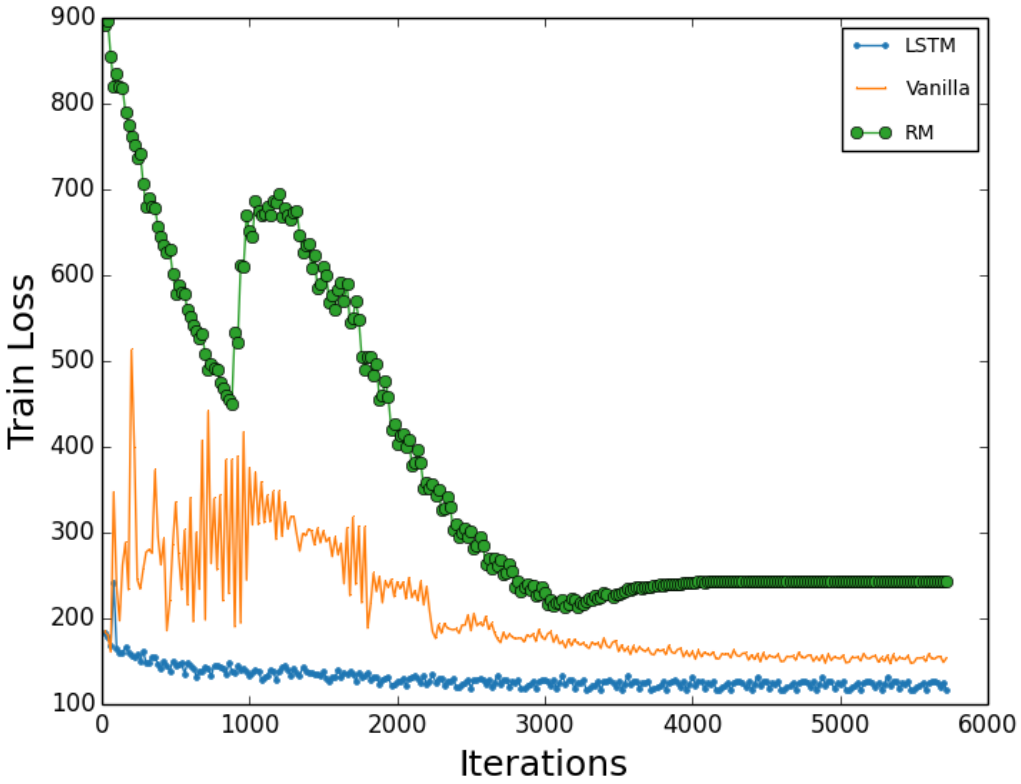


Figure 4.8: Training Loss vs. Iterations

I present the results on experiments with its analysis. Experiment is conducted on the Penn Tree Bank (PTB) dataset with evaluation on a word-level language model. I have compared RM with two baselines - vanilla RNN and RNN with LSTM units. Fig. 4.8 shows the training loss vs. iterations. The RNN network has 2 hidden layers with a embedding layer, and rolled out to 20 time steps. Each layer consists of 200 units. Training of all algorithms are using a batch size of 20 and vocabulary size of 10000. Training loss in the figure is represented by the perplexity on training data. For BPTT on vanilla RNN and LSTM RNN, they are trained with a learning rate of 1 with decay rate of 0.5.

4.2.2 Analysis And Improvements

We see that LSTM RNN has outperformed Vanilla RNN in terms of both convergence rate and converged loss, which is expected. RM, on the other hand, shows clear convergence but does not converge as fast as the baselines. Meanwhile, its converged loss is also higher than the baselines.

The convergence of RM shows that the algorithm is effective in RNN training, beyond the level of theory. However, further thought should be put into the design of the RDLG as it is not reaching the performance of vanilla RNN. In fact, the intrinsic problem of gradient vanishing might be hurting RM more than vanilla RNNs as it has reached better performance than back-propagation on feedforward networks [SZ16], meaning faster convergence rate.

I propose future work of revisiting the design of RDLG , taking into account the parameter sharing of RNN itself in the definition of utilities. A straightforward way is to introduce an equivalent game for LSTM RNN itself, instead of a plain RNN framework. Considering the recent rising trend of attention mechanism and transformers [VSP+17], it would be interesting to think of designing a game that is equivalent to the transformer architecture, instead of LSTMs. Also, modifications can be applied on RM to resolve the problem. Meanwhile, we could consider a constrained problem which is common in practice. Specifically, when establishing an RNN model, regularizations are usually enforced to boost the performance, to name a few popular ones: dropout [SHK+14], weight decay [KH92], and gradient clipping [PMB13]. For example, there is relation between dropout and pruning in games, as they both aim to reduce the parameter/action space.

4.3 Adversarial Learning

RDLG tries to boost convergence of optimization problems via an adversarial setting. RDLG takes a rather direct approach of designing a game from scratch, creating players, actions and utilities that are equivalent to their counterparts in the neural network architecture. Actually, the idea of combining game theory with machine learning has been expanding in recent research and there are more subtle ways of employing it.

Generative Adversarial Networks (or GANs) [GPAM$^+$14] stand out as an example of injecting game theory in deep learning. Rather than learning patterns of sample data, the GAN framework aims to jump out of the sample space and learn the actual distribution. In a zero-sum game, two networks are set up as the players against each other: a generative network generates fake samples from a latent space (usually random noise) while a discriminative network tries to distinguish it from real samples, which is essentially distinguishing between two distributions. Hence the objective/utility of the generative network is to increase the error rate of the discriminative network, while that of the dicriminative network is to minimize it. GANs are first applied to generate photorealistic images, and later on reconstructing 3D models from images and modling videos [VPT16]. It has been widely used in industry for all kinds of modifications on images [SEB$^+$18, ABD17]. The power of a zero-sum game and the nature of two opponent networks has enabled deep learning to achieve tasks that are impossible for other frameworks.

It is also worthwhile to point out the relation to unsupervised learning. GAN is a perfect example of combining both generative and discriminative models. In the design of a discriminative network, it remains a question of what constitutes a good

metric to evaluate difference between distributions [ZLPS17]. This falls into the category of questions on designing a comprehensive objective in an unsupervised task, which deserves a lot of thinking.

Game theory has provided a natural adversarial setting, which is helpful for a lot of problems in unsupervised learning. It has shown power of improving convergence and learning distributions, and there is undoubtedly plenty to explore [FCAL16].

CHAPTER V

Power of Unstructured Data

In previous chapters I have discussed multiple unsupervised models on various type of structured data. In this chapter I switch gears to unstructured data, specifically, text data, and show its power in unsupervised learning.

Approaching unstructured data is a significant part in machine learning, for structured data only takes a small proportion in the real world, and there is significant need in extracting information in unstructured data. Taking the view of data collection, structured data are compressed into a predefined form from unstructured data, which inevitably leads to information loss. On the other hand, unstructured data is not as difficult to learn for humans. For example, current models for syntactic annotation or parsing is way slower and less accurate than human, many natural language generation (or NLG) tasks such as machine translation and dialogue generation is far below human level. Therefore, there is no doubt that unexplored space of learning from unstructured data is still large.

I will discuss my study on unstructured data, specifically text data, in the context of representation learning. Learning word/sentence representations is the key to the success of NLP models, as they are the input to all the supervised classification/regression models. In fact, researchers have spent decades on finding the best

way to represent words in text data.

For text data, there exists a standard way for learning its representations. Usually, for a specific corpus, raw text data is preprocessed with a tokenizer that can operate on different levels. For smaller corpora we usually apply word-level tokenization, and a character-level tokenization can help restrict mapping to a limited but complete space. Byte Pair Encoding (or BPE) [SHB15] is a encoding method that has gained popularity recently. It creates a tokenizer adaptively to a corpus, with the key idea of encoding word level inputs for frequent symbols and character level inputs for less frequent ones. With a vocabulary predefined, text will be mapped to more structured tokens, and words/characters outside the vocabulary will all be replaced as the $< UNK >$ token. At this stage the "raw" one-hot embedding is created, where each token is represented by a unit vector of size $size(vocabulary)$, and the 1 entry lies in the mapped token.

The one-hot embeddings are then taken as the input to all unsupervised models and mapped to a much denser distributed representation of much smaller dimension. To learn a generic word embedding that ideally fits to multiple downstream tasks, there are two major types of models. One intuitive approach is to explore the task space. By training a embedding that simultaneously fits on multiple supervised tasks, it is more likely to obtain a representation with better generalization ability. This also alleviate the problem of small size of labeled data for most tasks and leverage multiple datasets. The other approach is within the unsupervised domain, creating tasks that are general enough to extract a representation that performs well on multiple tasks intrinsically. Autodencoders [Doe16], and language models [PNI+18, DCLT18, RNSS18] both fall into the latter. Recent works show that

there is signigicant improvement by combining these two types of models [LHCG19]. In this chapter, I focus on the unsupervised models.

5.1 Learning Generic Embeddings For Entities

In this study, I propose to extract generic embeddings for certain entities using unsupervised generic embedding learning methods. Embeddings refers to the representation of entities in a latent space using real-valued vectors. They are essential in the production of all kinds of tasks. For example, in the ride-sharing service, weather embeddings help forecast demand, road-segment embeddings are used to predict time of arrival [WFY18], and the customer-service chatbot's question answering(QA) engine takes words in a user's utterance as input features.

In this study, we consider user embeddings. That is, to extract generic representations of users from user history and profiles. Both drivers and passengers play important roles in all kinds of tasks - prediction of drunk drivers/passengers, prediction of passengers with children so appropriate service can be provided, and possibility of conflict between certain pairs. A pre-computed, published, low-dimensional user embedding would be beneficial to the entire service. This is a challenging task due to the sparsity of user data. In the ride-sharing scenario, it includes specific information such as trip information and words used in the user's calls to customer service, as well as meta data such as number of trips users take between certain hours.

Our contributions in this study are:

- We propose a framework to evaluate unsupervised entity embeddings in the ride-share service domain.
- We provide analysis on the user embeddings in an interpretable way.

- We propose an approach to leverage a heterogeneous dataset for user embedding extraction that is applicable to real data in the ride-share service domain.

5.1.1 Problem Formulation

The motivations of our study are:

- Extract generic user embeddings that best explain the full range of heterogeneous data involving those users.
- Extraction is done unsupervisedly, without presupposing any particular task for those embeddings.
- In order to evaluate the performance of different NLP embedding models, we start with synthetic data to gain understanding.

We define the problem as follows.

Problem 5.1. We assign "known" prior information to 5 passengers and 5 drivers. Each user will have a binary label assigned for his/her aggressiveness, workday pattern, propensity to complain, etc. We build a simulator that generates daily events of users, activities include and not limited to: pick-ups, drop-offs, customer service calls, etc. The logic of the simulator are detailed in Algorithm 6. Given a large event log generated by the simulator, we use generic embedding learning model to extract the embeddings of users, and then evaluate the embeddings.

5.1.2 Experiment Setup

The workflow of our experiment is shown in Fig. 5.1.

In our experiment, we predefine the profiles of 5 drivers and 5 passengers:

$$\text{Passenger0} = \{MILD, RELIABLE, WORKDAY\}$$

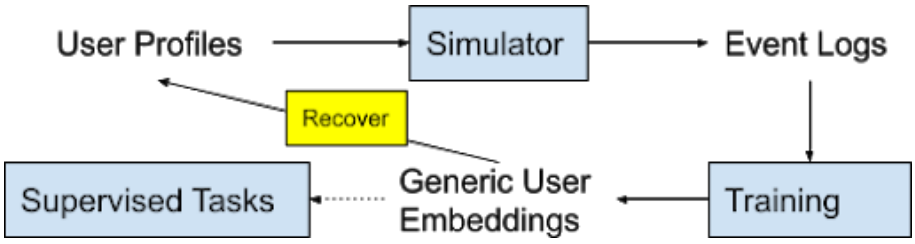


Figure 5.1: Workflow of experiment on user embedding extraction.

$$\text{Passenger1} = \{AGGRO, RELIABLE, MISCDAY\}$$

$$\text{Passenger2} = \{AGGRO, RELIABLE, WORKDAY\}$$

$$\text{Passenger3} = \{MILD, UNRELIABLE, MISCDAY\}$$

$$\text{Passenger4} = \{AGGRO, UNRELIABLE, MISCDAY\}$$

$$\text{Driver0} = \{MILD, RELIABLE, COMPLAINER\}$$

$$\text{Driver1} = \{AGGRO, RELIABLE, COMPLAINER\}$$

$$\text{Driver2} = \{MILD, UNRELIABLE, STOIC\}$$

$$\text{Driver3} = \{AGGRO, RELIABLE, STOIC\}$$

$$\text{Driver4} = \{MILD, UNRELIABLE, STOIC\}$$

We have generated logs spanning range of $D = \{100, 200, 400, 800, 1600, 3200, 6400, 12800\}$ as input of embedding models. A sample log is given as below:

Passenger2 called Driver3 at hour 1

Driver3 picked up Passenger2 at hour 1

Driver3 argued with Passenger2 at hour 1

Passenger0 called Driver2 at hour 2

Driver2 and Passenger0 never met at hour 2

Passenger1 called Driver4 at hour 2

Driver4 picked up Passenger1 at hour 2

Driver4 argued with Passenger1 at hour 2

Passenger2 called Driver0 at hour 2

Driver0 picked up Passenger2 at hour 2

...

5.1.3 Methods

We apply 4 training methods to the event logs: word2vec [MCCD13], LSTM-based language model [ZSV14], autoencoder [Doe16], and node2vec [GL16]. The former 3 methods are applied directly on the text data, while node2vec is applied on a graph generated from samples on text data, as we describe in detail below. Although there are many other updated and sophisticated methods for extracting embeddings, we choose these 4 methods as they are either the foundations of word embeddings or represent a classical type of unsupervised tasks for text data. We give the details of our implementations as follows. The hyper-parameters are chosen either by the default setting or in experiments where the original training sets are comparable to ours.

- For word2vec, we choose a window size of 5, and number of negative samples for NCE (noise contrastive estimation) as 25, a learning rate of 0.025, and a

data size $D = 1600$.

- For node2vec, we create a directed, weighted co-occurrence graph from the log corpus, where each node represents a unique word, the direction represents one word following another immediately in the text, and the edge weights represent the frequency of the word-pair co-occurrences.
- The language model consists of 2 hidden layers, and trained with an initial learning rate of 1 and a decay rate of 0.5 for epochs ≥ 13. Each layer is a stack of LSTM units of dimension size/hidden size specified beforehand.
- In a bidirectional RNN-based autoencoder, the number of LSTM units is chosen to be 500, and learning rate of 0.001. As it is a rather complex model, number of maximum epochs is set to 4.

For simplicity both on the data itself and for evaluation, we learn embeddings of size 1, 2, and 4. For embeddings of size 4, we project the embedding to 2D for visualization using t-SNE [MH08], which is entirely independent on our extraction methods.

5.1.4 Evaluation

The purpose of learning 1D embeddings is to see the power of models under extreme cases. If there is only 1 unit to use to represent a word, what can it possibly represent for? What can it be used to distinguish? For all 4 different models we have given 5 runs and looked at the 1D embeddings for both regular words and user words (i.e. “Passenger0”, “Driver1”, etc.). Observations have shown no agreement on the capability of the 1D embeddings, in terms of either distinguishing the users/regular words or passengers/drivers. This result shows that the power of 1D embeddings is

still quite weak, providing no interpret ability on the data. We can also categorize the observation as a classical result of underfitting - 1D is simply not enough to adapt to the information in the data. 1D embeddings serve as a good unit test for evaluations on the capacity as well as the interpretability of these embedding models.

Moving on to 2D embedding learned, we see all 4 models demonstrating the power to distinguish different type of users. Visuallization of the embeddings shows that the embeddings are capable of revealing the user profiles in an interpretable way. Specifically, for each feature of passengers/drivers, there is a hard boundary we can find to separate it, demonstrating the power of generic embeddings in distincting different user from different perspectives. We show the embeddings of these models of word2vec and language model in Fig. 5.2 and Fig. 5.3. The orange lines show the clear separation of features. For example, MILD/AGGRO drivers are clearly separated, and RELIABLE/UNRELIABLE passengers are separated in every trial. Node2vec and autoencoders demonstrate the same level of power of distinguishing different type of users.

We have also experimented on 4D embeddings. And then project them to 2D using t-SNE. Even though the objective of t-SNE is independent from our models, we are still able to obtain similar results for some trials. However, also because t-SNE does not follow the objective of the models, meaning it does not guarantee to find the optimal separation plane, it is inevitably difficult to find hard boundaries to separate the users.

It is worthwhile to point out that we learned the embeddings without any prior information of either distinction of users from other words in the vocabulary, nor any context information of words from pretrained embeddings. With the intepretable em-

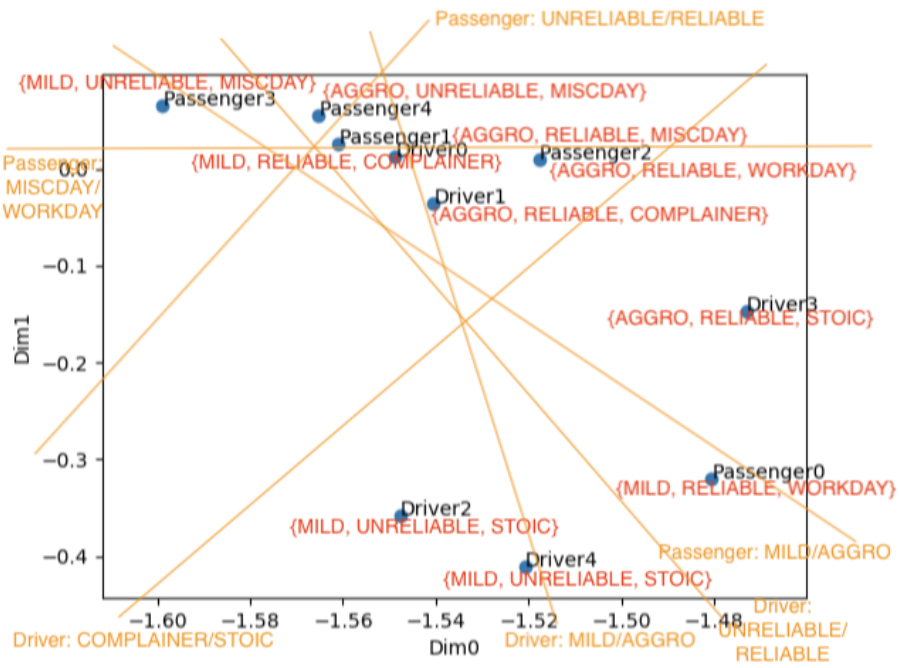


Figure 5.2: Word2vec: 2D embedding. Each profile feature is clearly distinguished by the embedding.

beddings, there is strong evidence to believe the generic user embeddings have great potential in performing well in a set of downstream tasks. In fact, this has been demonstrated in recent works. In the latest GPT2 model, OpenAI has demonstrated the ability of embeddings learned from language models in performing multiple downstream tasks in a zero-shot setting, taking advantage of a giant heterogeneous corpus WebText crawled from the web.

Our workflow in Fig III will find huge potential in industry as data is affluent. In a ride-sharing company, there are sets of heterogeneous data involving all kinds of user activities: user profiles, trip information, users' dialogs etc. A model that can take all data sources as input and generate user embeddings will be extremely helpful for all kinds of downstream tasks. Considering the different formats of data, we propose to convert all structured data into unstructured, and feed it to the embedding models. This is not only easy to implement, and also will serve as a good

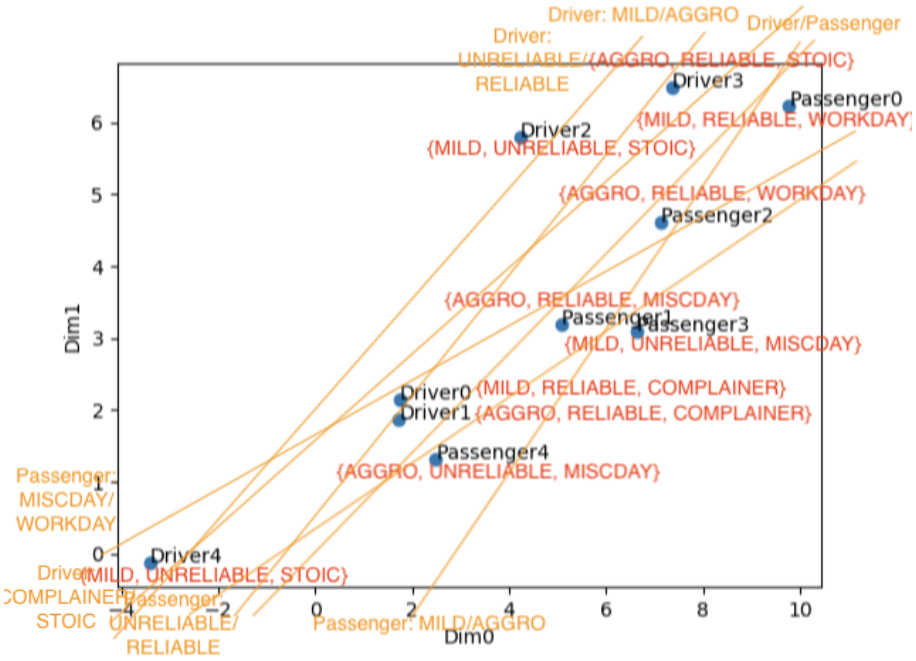


Figure 5.3: Language model: 2D embedding. Each profile feature is clearly distinguished by the embedding.

evaluation of power of unstructured data. Thinking beyond the users, our workflow is capable of extracting embeddings of any type of entities. For instance, given a text log on trip information, POI (point of interest) emebddings can be extracted for many prediction tasks depending on traffic conditions. Considering the scale of real data and the lack of labels, we generate embeddings of dimension 200 to 300, and we do not perform the evaluations that we did on synthetic data. Instead, embeddings are directly used by downstream tasks for evaluation. Due to data confidentiality, we do not show the experimental results on real data.

Algorithm 6 User Activity Simulator

Set number of days D, number of drivers n, number of passengers m, profiles of $\{driver0, \ldots, drivern\}$ and $\{passenger0, \ldots, passengerm\}$

function CALL CAR(passengerId)
    Choose an unoccupied driver $driverId$ at random// driver who has not picked up a passenger in current loop
    Output to log: "passengerId called driverId at hour h
    if driver and passenger both RELIABLE **then**
        Successful pick-up with $p = 0.95$
    else if driver is RELIABLE or passenger is RELIABLE **then**
        Successful pick-up with $p = 0.8$
    else// neither is RELIABLE
        Successful pick-up with $p = 0.6$
    end if
    if pick-up successful **then**
        Output to log: "driverId picked up passengerId at hour h"
        if driver and passenger both AGGRO **then**
            Argue with $p = 0.95$
        else if driver or passenger AGGRO **then**
            Argue with $p = 0.5$
        else
            Argue with $p = 0.1$
            if Argue **then**
                Output to log: "driverId argued with passengerId at hour h"
                if driver is COMPLAINER **then**
                    Output to log with $p = 0.8$: "driverId complains to customer service"
                end if
            end if
        end if
    else// pick-up unsuccessful
        Output to log: "driverId and passengerId never met at hour h"
        if driver is COMPLAINER **then**
            Output to log with $p = 0.3$: "driverId complains to customer service"
        else// driver is STOIC
            Output to log with $p = 0.1$: "driverId complains to customer service"
        end if
    end if
end function

for $d = 1, \ldots, D$ **do**
    for $h = 00, \ldots, 23$ **do**
        for $passengerId = 0, \ldots, m$ **do**
            if passenger is WORKDAY **then**
                if $h = 08$ or $h = 18$ **then**
                    CALL CAR(passengerId) with $p = 0.9$
                else
                    CALL CAR(passengerId) with $p = 0.05$
                end if
            else// passenger is MISCDAY
                CALL CAR(passengerId) with $p = 0.1$ // regardless of h
            end if
        end for
    end for
end for

CHAPTER VI

Conclusion

Attention on unsupervised learning has boosted significantly for the last decade, and is very like to keep climbing due to its considerable potential in the big data era. In this thesis, I give an overview of unsupervised learning from different perspectives. Following the taxonomy on data types, I discuss my contributions to unsupervised learning on graphs/networks and sequences individually in detail. Specifically, I introduce CondeNSe, an algorithm that summarizes a large graph into a succinct supergraph in scale, and the correlation between graph summarization and graph clustering. As for sequences I have reviewed popular research problems and proposed a problem CCTN that cluster time-series and graphs jointly, with its solutions and applications. I also revisit the classical problem of sequence prediction and introduce a novel game-theoretic algorithm RDLG to solve it. Extending the game design I briefly talk about its relation to adversarial learning. Beyond structured data such as networks and sequences, I additionally demonstrate the power of unsupervised learning applied on unstructured data, showing by experimentation that present unsupervised representation learning models have tremendous potential in solving for multiple tasks and helping interpret text data. Besides the contribution of novel algorithms on rich data types, we have been able to see the flexibility of unsupervised

methods on learning patterns in complex data sets, as well as its power to preprocess data and serve for all kinds of downstream tasks. For different and more types of data, objectives need to be carefully designed to extract the desired patterns. And for well-established objectives, optimization methods can be further improved by taking advantage of an adversarial setting. The generalization ability and interpretability of many unsupervised models, especially on unstructured data remain unclear, but there are ways to can evaluate them in specific application scenarios.

There is vast space to explore from various perspectives. Embeddings for networks is an important topic in representation learning and has been studied for years [TQW$^+$15, PARS14, RSF17, CLX16, DZHL18, HSSK18, JRK$^+$18, JHRK19]. It has been shown that effective representations can be learned in unsupervised ways which deserves further exploration and understanding. Evaluations of unsupervised learning need a thorough look, is it interpretability or downstream task performance that we care about? It is worthwhile to think about, both theoretically and empirically, what constitutes a good objective for unsupervised learning tasks, and what makes an objective "better" than another? Representation learning is one of the most critical fields we focus on. As data volume grows and deep learning models gains popularity, what can we learn from a giant source of heterogeneous data, or unstructured data? These are the questions we need to dig deeper and find better understandings on. Nevertheless, throughout this thesis, I have demonstrated the efficacy of unsupervised learning in learning from data regardless of its type.

APPENDIX

.1 CCTN: Derivations and Proofs

In this section we provide the proofs for the two main results behind our proposed methods, CCTN and CCTN-INC.

Theorem 4.6. When fixing variable $\mathbf{C}$, we only need to focus on the parts of the objective (4.5) that depend on $\mathbf{W}$, so the objective becomes:

$$J(\mathbf{W}) = \sum_{k=1}^{K} a_k \|\mathbf{X}_k - \mathbf{C}\mathbf{W}\|_F^2 \tag{.1}$$

which corresponds to a quadratic optimization function. To optimize Eq. (.1), we apply the first order optimality condition as follows:

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}} = 2(\textstyle\sum_k a_k \mathbf{C}^T\mathbf{C})\mathbf{W})\mathbf{W} - 2\sum_k a_k \mathbf{C}^T\mathbf{X}_k = \mathbf{0}$$

Hence, $\mathbf{W}$ is the solution of linear equation (4.7).

When we fix $\mathbf{W}$, we need to take into account the parts of the objective function that depend on $\mathbf{C}$:

$$Q(\mathbf{C}) = \sum_{k=1}^{K} a_k \|\mathbf{X}_k - \mathbf{C}\mathbf{W}\|_F^2 + \lambda Tr(\mathbf{C}^T\mathbf{L}\mathbf{C}). \tag{.2}$$

To minimize $Q(\mathbf{C})$ with respect to $\mathbf{C}$, we take the partial derivative and set it to 0:

$$\frac{\partial Q(\mathbf{C})}{\partial \mathbf{C}} = 0 \Rightarrow 2\sum_k a_k(\mathbf{C}\mathbf{W} - \mathbf{X}_k)\mathbf{W}^T + 2\lambda\mathbf{L}\mathbf{C} = 0.$$

By rearranging this equation, we obtain Eq. (4.8). □

Thoerem 4.11. The embedded clustering matrix $\mathbf{C}'$ is fixed and initialized to the solution of the non-incremental version, i.e., $\mathbf{C}' = \mathbf{C}$. According to Theorem 4.6,

and using the notation that we introduced for perturbations, we have:

$$\mathbf{W}' = \left(\sum_k a_k \mathbf{C}'^T \mathbf{C}'\right)^{-1} \sum_k a_k \mathbf{C}'^T \mathbf{X}'_k \Rightarrow$$

$$\mathbf{W} + \Delta\mathbf{W} = \left(\sum_k a_k \mathbf{C}^T \mathbf{C}\right)^{-1} \sum_k a_k \mathbf{C}^T (\mathbf{X}_k + \Delta\mathbf{X}_k)$$

By using the formula for $\mathbf{W}$ from Theorem 4.6 and substituting it above, we obtain the expression in Eq. (4.12). Then, we apply Theorem 4.6 to find the 'ground truth' solution of $\mathbf{C}'$ (based on CondeNSe), when $\mathbf{W}'$ is fixed:

$$\mathbf{C}' \textstyle\sum_k a_k \mathbf{W}' \mathbf{W}'^T + \lambda \mathbf{L}' \mathbf{C}' = \sum_k a_k \mathbf{X}'_k \mathbf{W}'^T$$

By rewriting the last equation using the Δ-notation for all the matrices that are involved, we obtain

$$(\mathbf{C} + \Delta\mathbf{C}) \sum_k a_k (\mathbf{W} + \Delta\mathbf{W})(\mathbf{W} + \Delta\mathbf{W})^T$$
$$+ \lambda(\mathbf{L} + \Delta\mathbf{L})(\mathbf{C} + \Delta\mathbf{C}) = \sum_k a_k (\mathbf{X}_k + \Delta\mathbf{X}_k)(\mathbf{W} + \Delta\mathbf{W})^T$$

Based on the assumption that the perturbations are all significantly sparser than the original matrices, we can approximate the above equation by ignoring their 2^{nd}-order terms. That is, we drop the terms of the form $(\Delta\mathbf{M})^2$, where $\mathbf{M} \in \{\mathbf{C}, \mathbf{W}, \mathbf{L}, \mathbf{X}_k\}$. By expanding the last equation and applying this matrix approximation, we obtain Eq. (4.13). □

.2 CCTN-inc: Incremental Algorithm

In Sec. 4.1, we leveraged Theorem 4.11 to propose CCTN-inc, an effective and fast approximation of CCTN, which handles incremental updates in the network

structure, the introduction of new nodes, and changes in the behavioral patterns of existing nodes. In Algorithm 7, we give the high-level pseudocode of CCTN-INC.

Algorithm 7 CCTN-INC: Incremental version of CONDENSE

Input: Graph $G'(\mathcal{V}', \mathbf{E}')$; stacked matrices of k-type time-series $\{\mathbf{X}'_k\}$ with T' timesteps, basis matrix of temporal patterns $\mathbf{W}$ and embedded-clustering matrix $\mathbf{C}$ found by Alg. 3; parameters a_k and λ, dimensionality d (same as Alg. 3)
Output: Vector with cluster assignments $\mathbf{c}'$

$\epsilon = 10^{-6}, \tau_{max} = 100$ // Constants for convergence
$\tau = 0$ // Initialization of iteration #
// Initialization based on the solutions of Alg. 3
$\mathbf{C}'_{(\tau)} = \mathbf{C}$ // Random init of rows of new nodes
$\mathbf{W}'_{(\tau)} = \mathbf{W}$ // Random init of cols of new timestamps
repeat
  $\tau = \tau + 1$
  // **Step 1**: Compute $\mathbf{W}'$ via Eq. (4.12)
  $\mathbf{W}'_{(\tau)} = (\sum_k a_k \mathbf{C}'^T_{(\tau-1)} \mathbf{C}')^{-1} (\sum_k a_k \mathbf{C}'^T_{(\tau-1)} \mathbf{X}'_k)$
  // **Step 2**: Compute $\mathbf{C}'$ via Eq. (4.13)

$$\Delta\mathbf{C}_{(\tau)} \sum_k a_k \mathbf{W}_{(\tau)} \mathbf{W}^T_{(\tau)} + \lambda \mathbf{L} \Delta\mathbf{C} = \sum_k a_k (\mathbf{X}_k (\Delta\mathbf{W}_{(\tau)})^T + (\Delta\mathbf{X}_k)\mathbf{W}^T_{(\tau)} - \mathbf{C}(\Delta\mathbf{W}_{(\tau)})\mathbf{W}^T_{(\tau)} - \mathbf{C}_{(\tau)}\mathbf{W}_{(\tau)}(\Delta\mathbf{W}_{(\tau)})^T + \lambda(\Delta\mathbf{L})\mathbf{C}_{(\tau)})$$

until $(||\mathbf{C}'_{(\tau)} - \mathbf{C}'_{(\tau-1)}||_1 < \epsilon$ & $||\mathbf{W}'_{(\tau)} - \mathbf{W}'_{(\tau-1)}||_1 < \epsilon)$ or $\tau > \tau_{max}$
// **Step 3**: Assign the nodes to clusters based on the inferred embeddings in **C**' (each row is an 'observation').
$\mathbf{c}' = \text{cluster\_rows}(\mathbf{C}'_{(\tau)})$
return $\mathbf{c}'$

.3 CCTN: Baselines

Here we present some details about the clustering methods that we considered as baselines in our experiments.

• **Spectral clustering** refers to a class of algorithms that utilize eigendecomposition to identify community structure. We utilize one such algorithm [Hes04], which partitions a graph by performing k-means clustering on the top-k eigenvectors of the input graph. The idea behind this clustering is that nodes with similar connectivity have similar eigen-scores in the top-k vectors, and thus form clusters.

• **Louvain** is a modularity-based partitioning method for detecting hierarchical community structure. The method is iterative: (i) Each node is placed in its own community. Then, the neighbors j of each node i are considered, and i is moved to j's community if the move produces the maximum modularity gain. The process is applied repeatedly until no further gain is possible. (ii) A new graph is built whose supernodes represent communities, and superedges are weighted by the weighted sum of links between the two communities. The algorithm typically converges after a few passes.

• **k-Shape** relies on a scalable iterative refinement procedure, which creates homogeneous and well-separated clusters. As for the distance measure, k-Shape uses a normalized version of the cross-correlation in order to consider the shapes of time series while comparing them. Based on the properties of this distance measure, a method is developed to compute cluster centroids, which is used in every iteration to update the assignment of time series to clusters.

• Based on DTW, **TADPole** uses a pruning strategy that exploits both upper and lower bounds to remove a large fraction of the expensive distance calculations. This pruning strategy gives provably identical results to the brute force algorithm, but is at least an order of magnitude faster. It also uses a simple heuristic to order the calculations, thus casting the clustering as an anytime algorithm.

We discuss the parameter settings for the baselines in Sec. 4.1, and the setup for this experiment in Sec. 4.1.

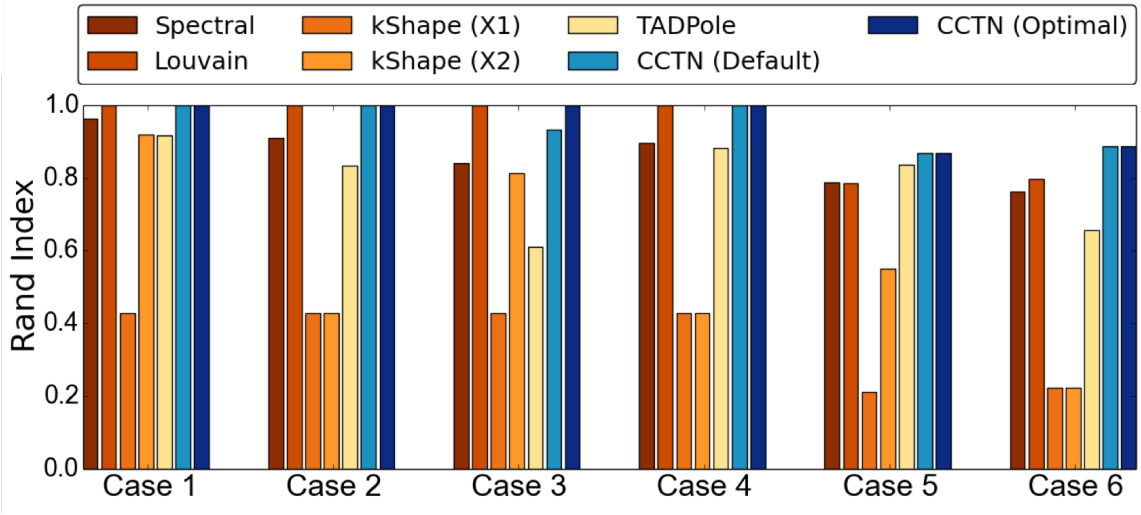
.4 CCTN: Accuracy of Cluster Recovery

In Sec. 4.1, we presented the results from comparing our method, CCTN, with the baselines in terms of accuracy (i.e., agreement with the ground-truth clusters in the labeled data). In the main paper, we included the plots that compare the methods in terms of NMI. Here we supplement our analysis with results on the rand index in Fig. A.1. The trends are similar to the ones we see for NMI, though the scores are consistently higher. Overall, CCTN has consistently strong performance across a wide variety of synthetic and real datasets, unlike the baselines which exhibit wide variability in their performance—for example, k-Shape works well on MITRE in terms of the rand index, but performs poorly or worse than CCTN in the other real datasets and many of the synthetic cases.

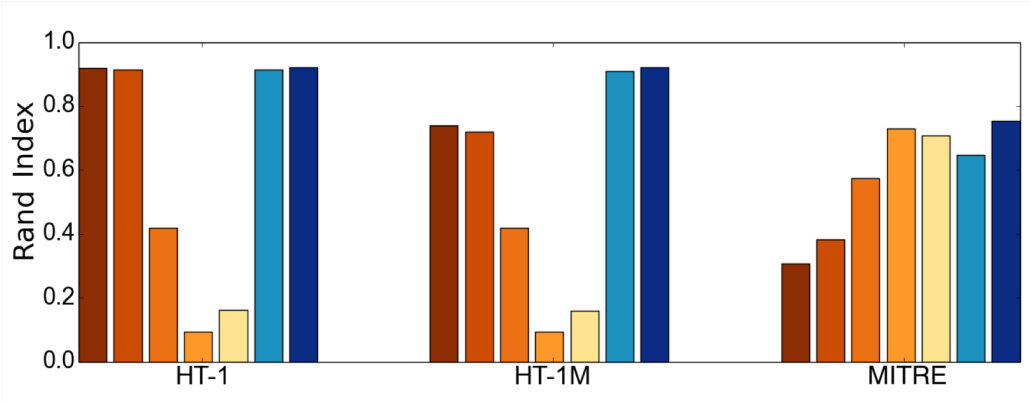
In Fig. A.2, we also present rand-index based results about the approximation of CCTN by its incremental counterpart, CCTN-INC. The setup for this experiment is given in Sec. 4.1(2). We observe that, based on both metrics, CCTN-INC approximates CCTN very well in that it finds similar clusterings. The trends between NMI and rand index are similar, with rand index being consistently higher.

.5 CCTN: Sensitivity of Clustering

The last question that we raised in Experiments of Sec. 4.1 is about the robustness of CCTN to different parameter settings. To answer this question, we experiment with different combinations of parameters and evaluate the corresponding effect on clustering. In this experiment we call $a_1 = a_2 = \lambda = 1, d = 4$ 'default setting'. Note that these values are different from the default case that we discuss in Sec. 4.1. Then, we run CCTN on all the synthetic cases of Table 4.2 by fixing all the parameters to



(a) Synthetic data: CCTN performs better than the baselines on difficult cases or equally well on easy cases.



(b) Real data: Overall, CCTN performs consistently better than or equally well as the baseline methods.

Figure A.1: Rand index-based accuracy of CCTN and CCTN-INC on synthetic and real data.

their default setting, with the exception of one parameter that we vary. The results are shown in Figure A.3.

Observation 5. CONDENSE is generally robust to different parameter settings. Based on the rand index, it is more sensitive to λ than the other parameters. Practitioners may choose to tune λ primarily to achieve better performance.

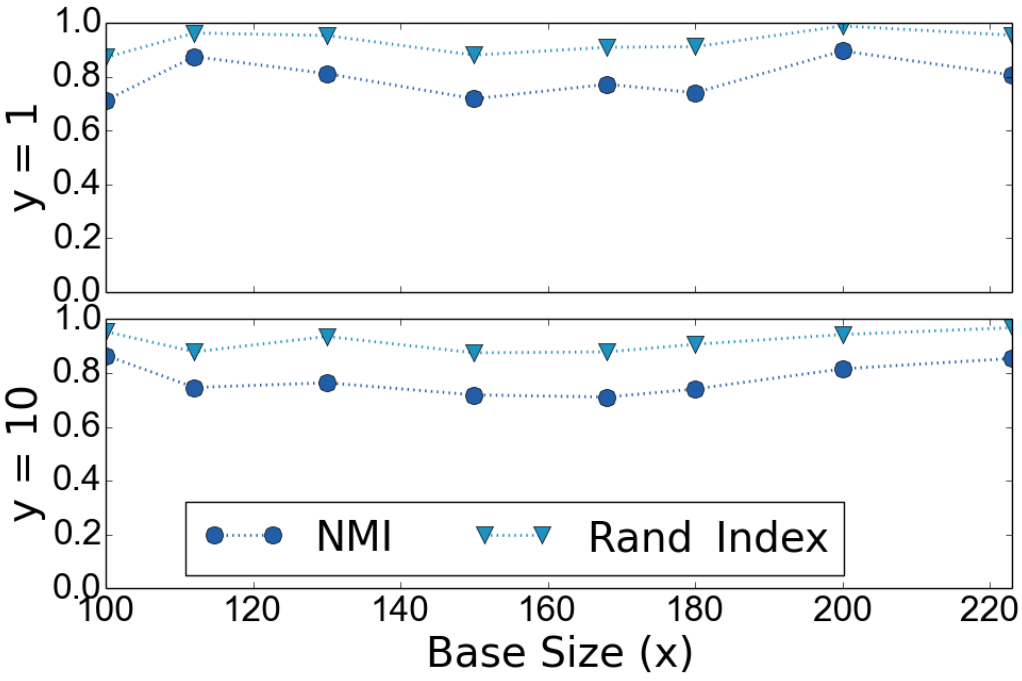
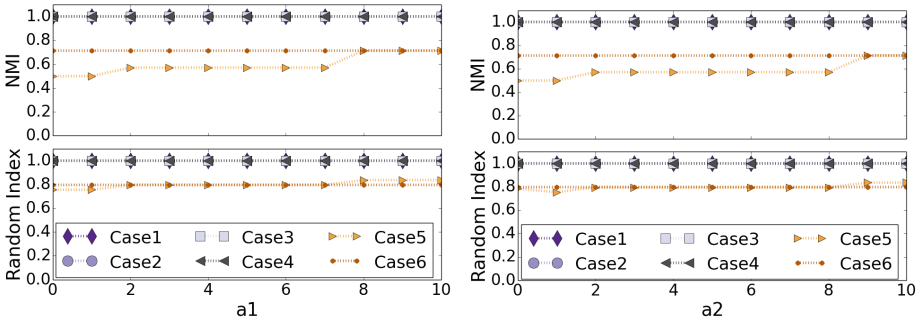
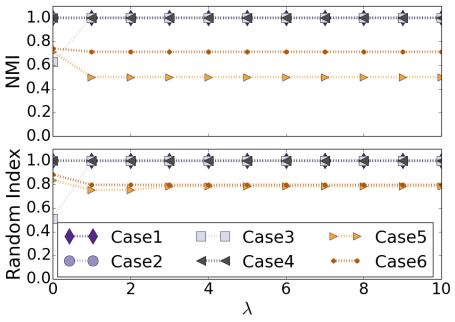


Figure A.2: MITRE: CCTN-INC approximates CCTN well based on both the NMI and rand index metrics.

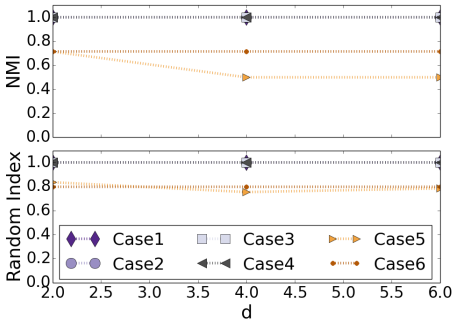


(a) Sensitivity to a_1.

(b) Sensitivity to a_2.



(c) Sensitivity to λ.



(d) Sensitivity to d.

Figure A.3: CCTN is robust with respect to the parameters. The different lines in the plots correspond to different synthetic cases, as described in the rightmost plot (d).

.6 RDLG : Proofs

.6.1 Group of Nodes

For the RDLP , bounded constraints are enforced on groups of nodes, so we can exploit the partition trick as in [SZ16] for nodes with same constraints when we design the game.

.6.2 Reasonable Actions & Nash Equilibria

We define reasonable actions for recurrent deep learning games. Proof of Nash equilibria is similar to that of feedforward networks [SZ16].

Definition 0.3. Given a joint action for the recurrent deep learning game $\mathbf{a} = (\mathbf{U}, \mathbf{V}, \mathbf{W}, \{\mathbf{a}^{(t)}, b^{(t)}\}, \{q_v^{(t)}, d_v^{(t)}\})$ and some $v \in x \cup h \cup o$, if f_v for $v \in x \cup h$ or g_v for $v \in o$, is convex and differentiable, define the zanni at v to be reasonable for $\mathbf{a}$ if for all $t \in \{1, \ldots, \tau\}$:

for $v \in x$, $q_v^{(t)} = 1$, and $c^{(t)}(v, \mathbf{U}, \mathbf{V}, \mathbf{W} = d_v^{(t)} + q_v^{(t)} c^{(t)}(v, \mathbf{U}, \mathbf{V}, \mathbf{W})$;

for $v \in h$,

$$
\begin{aligned}
& q_v^{(t)} \\
& = f_v'\Big(\sum_{u:(u,v)\in E} c^{(t-1)}(u, \mathbf{U}, \mathbf{V}, \mathbf{W}) W(u, v) + \sum_{u':(u',v)\in E} c^{(t)}(u', \mathbf{U}, \mathbf{V}, \mathbf{W}) U(u', v)\Big),
\end{aligned}
$$

and

$$
\begin{aligned}
& f_v\Big(\sum_{u:(u,v)\in E} c^{(t-1)}(u, \mathbf{U}, \mathbf{V}, \mathbf{W}) W(u, v) + \sum_{u':(u',v)\in E} c^{(t)}(u', \mathbf{U}, \mathbf{V}, \mathbf{W}) U(u', v)\Big) \\
& = d_v^{(t)} + q_v^{(t)}\Big(\sum_{u:(u,v)\in E} c^{(t-1)}(u, \mathbf{U}, \mathbf{V}, \mathbf{W}) W(u, v) \\
& + \sum_{u':(u',v)\in E} c^{(t)}(u', \mathbf{U}, \mathbf{V}, \mathbf{W}) U(u', v)\Big);
\end{aligned}
$$

for $v \in o$,

$$q_v^{(t)} = g_v'(\sum_{u:(u,v)\in E} c^{(t)}(u, \mathbf{U}, \mathbf{V}, \mathbf{W})V(u, v));$$

and

$$g_v(\sum_{u:(u,v)\in E} c^{(t)}(u, \mathbf{U}, \mathbf{V}, \mathbf{W})V(u, v))$$
$$= d_v^{(t)} + q_v^{(t)}(\sum_{u:(u,v)\in E} c^{(t)}(u, \mathbf{U}, \mathbf{V}, \mathbf{W})V(u, v)).$$

Definition 0.4. If the loss l is convex and differentiable, then the antagonist is reasonable if for all $t \in \{1, \ldots, \tau\}$, $\mathbf{a}^{(t)\intercal}\mathbf{c}^{(t)}(o, \mathbf{U}, \mathbf{V}, \mathbf{W}) + b^{(t)} = l^{(t)}(\mathbf{c}^{(t)}(o, \mathbf{U}, \mathbf{V}, \mathbf{W}))$ and $\mathbf{a}^{(t)} = \nabla l^{(t)}(\mathbf{z})|_{\mathbf{z}=\mathbf{c}^{(t)}(o,\mathbf{U},\mathbf{V},\mathbf{W})}$.

Proof of Lemma 4.20 is then analogous to Lemma 3 in [SZ16], with the definitions in Section .6 and above. Note that when generating the partial order, we consider the deep network without the recurrent edges, i.e. the edge from h to h removed from the example, which is a acyclic graph. We define partial order in this way for two reasons: only acyclic graphs can generate partial order; the recurrent edges connect values from time step t to $t + 1$, which is an order already in place because of the way we feed the data into the network and evaluate the network: input is a sequence $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\tau)}$ and we always finish evaluating one sample before the next, as shown in the rollout network in Figure 4.6. This partial order is the foundation of the proof that is built on top of strong induction.

Note that even though we have dependence among utilities of different t, the total utilities are still linear combinations of that of each t, hence the proof which break apart U^p into $U^{p(t)}$ remain valid.

The proof of Theorem 4.21 is also analogous to that of Theorem 4 in [SZ16].

We give the lemmas as follows.

Fact 0.5. Given a finite set S, a partial ordering $\leq$ over S, and a set $X \subseteq S$, then if for all $s' \in S$, $\{s' \in S : s' < s\} \subseteq X \Rightarrow s \in X$, then $X = S$.

Lemma 0.6. *Assume that for all $v \in x \cup h$ and $v' \in o$, f_v and $g_{v'}$ are convex and differentiable. Assume $\subseteq$ is the partial order generated by the directed acyclic graph in the recurrent deep network with the recurrent edges removed. For any $v \in x \cup h \cup o$, given a joint action $\mathbf{a} = (\mathbf{U}, \mathbf{V}, \mathbf{W}, \{\mathbf{a}^{(t)}, b^{(t)}\}, \{q_v^{(t)}, d_v^{(t)}\})$ where for all $u \leq v$, the zanni at u is reasonable for $\mathbf{a}$, then $U_v^{s(t)} = c^{(t)}(v, \mathbf{U}, \mathbf{V}, \mathbf{W})$.*

Proof. Define $U \subseteq x \cup h \cup o$ to be the set of all vertices $u \in x \cup h \cup o$ where $u \leq v$. Define $R \subseteq U$ to be the set of nodes v where $U_v^{s(t)}(\mathbf{a}) = c^{(t)}(v, \mathbf{U}, \mathbf{V}, \mathbf{W})$. We can use the partial order of the graph as a partial order over U to prove recursively that $R = U$. Then we can prove by strong strong recursion on this total order that $U_u^{s(t)}(\mathbf{a}) = c^{(t)}(u, \mathbf{U}, \mathbf{V}, \mathbf{W})$ if for all $u' < u$, $U_u^{s(t)}(\mathbf{a}) = c^{(t)}(u', \mathbf{U}, \mathbf{V}, \mathbf{W})$.

□

Lemma 0.7. *Assume that for all $v \in x \cup h$ and $v' \in o$, f_v and $g_{v'}$ are convex and differentiable, the loss l is convex and partially differentiable. Given a joint action $\mathbf{a} = (\mathbf{U}, \mathbf{V}, \mathbf{W}, \{\mathbf{a}^{(t)}, b^{(t)}\}, \{q_v^{(t)}, d_v^{(t)}\})$ where all zannis and the antagonist are reasonable, then for any example t in the sequence, $U^{a(t)}(\mathbf{a}) = l^{(t)}(c^{(t)}(v, \mathbf{U}, \mathbf{V}, \mathbf{W}))$.*

Proof. The proof is analogous to the proof of Lemma 0.6 □

Lemma 0.8. *Assume that for all $v \in x \cup h$ and $v' \in o$, f_v and $g_{v'}$ are convex and differentiable, the loss l is convex and partially differentiable. Given a joint action $\mathbf{a} = (\mathbf{U}, \mathbf{V}, \mathbf{W}, \{\mathbf{a}^{(t)}, b^{(t)}\}, \{q_v^{(t)}, d_v^{(t)}\})$ where all zannis are reasonable, then the unique best response for the antagonist is to be reasonable.*

Proof. Since the zanni knows the example, fix a specific example t. Define $z = x_v^{(t)}$, if $v \in x$, or □

Lemma 0.9. *Assume that for all $v \in x \cup h$ and $v' \in o$, f_v and $g_{v'}$ are convex and differentiable, the loss l is convex and partially differentiable. Given a joint action $\mathbf{a} = (\mathbf{U}, \mathbf{V}, \mathbf{W}, \{\mathbf{a}^{(t)}, b^{(t)}\}, \{q_v^{(t)}, d_v^{(t)}\})$ where all zannis are reasonable, and the antagonist is reasonable, the protagonists play $(\mathbf{U}, \mathbf{V}, \mathbf{W})$, then if U^p is the utility of the protagonists, then: $\nabla_{\mathbf{M}} U^p(\mathbf{a}) = -\nabla_{\mathbf{M}} L(\mathbf{U}, \mathbf{V}, \mathbf{W})$, where $\mathbf{M} \in \{\mathbf{U}, \mathbf{V}, \mathbf{W}\}$.*

For RNN we define paths between nodes on the network with the recurrent edges removed, i.e. if N' is the network with recurrent edges removed from network N, define $P(u, v)$ to be the set of all paths from u to v. For any path ρ, define $|\rho|$ to be the number of nodes in the path. For all $\rho \in P(u, v)$, $\rho_1 = u$ and $\rho_{|\rho|} = v$. As long as we don't refer to specific t, there is no ambiguity in this definition in RNNs.

Lemma 0.10. *for $v \in h$,*

$$\frac{\partial U^p(\mathbf{a})}{\partial W(u,v)} = -\frac{1}{\tau} \sum_{t=2}^{\tau} \sum_{k=1}^{n} a_k^{(t)} U_u^{s(t-1)}(\mathbf{a}) q_{\rho_{|\rho|}}^{(t)} a_k^{(t)} \Pi_{j=1}^{|\rho|-1} M(\rho_j, \rho_{(j+1)}) q_{\rho_{(j+1)}}^{(t)},$$

$$\frac{\partial U^p(\mathbf{a})}{\partial U(u,v)} = -\frac{1}{\tau} \sum_{t=1}^{\tau} \sum_{k=1}^{n} a_k^{(t)} U_u^{s(t)}(\mathbf{a}) q_{\rho_{|\rho|}}^{(t)} a_k^{(t)} \Pi_{j=1}^{|\rho|-1} M(\rho_j, \rho_{(j+1)}) q_{\rho_{(j+1)}}^{(t)};$$

for $v \in o$,

$$\frac{\partial U^p(\mathbf{a})}{\partial V(u,v)} = -\frac{1}{\tau} \sum_{t=1}^{\tau} \sum_{k=1}^{n} a_k^{(t)} U_u^{s(t)}(\mathbf{a}) q_{\rho_{|\rho|}}^{(t)} a_k^{(t)} \Pi_{j=1}^{|\rho|-1} M(\rho_j, \rho_{(j+1)}) q_{\rho_{(j+1)}}^{(t)};$$

where $M \in \{U, V, W\}$, depending on the position of v and which weight matrix the path involves. At the encountering of h, the recurrent edge is considered as extra input and included in the computation as the unfolded network suggests.

For each $\rho \in P$, define $U^p_{\rho,\mathbf{a}} : \mathbb{R}^{E_\rho} \Rightarrow \mathbb{R}$ such that $U^p_{\rho,a}(\mathbf{U}_\rho, \mathbf{V}_\rho, \mathbf{W}_\rho)$ is the utility of the protagonist at ρ if she unilaterally deviates from $\mathbf{a}$ to play $(\mathbf{U}_\rho, \mathbf{V}_\rho, \mathbf{W}_\rho)$.

The following is true for unconstrained RNNs. Proof not valid on constrained RNNs corresponding to KKT point.

Lemma 0.11. *Given a fixed protagonist action* $(\mathbf{U}, \mathbf{V}, \mathbf{W})$*, there exists a unique joint action for all agents* $\sigma = (\mathbf{U}, \mathbf{V}, \mathbf{W}, \{\mathbf{a}^{(t)}, b^{(t)}\}, \{q_v^{(t)}, d_v^{(t)}\})$ *where the zannis and the antagonist are playing best response to* σ*. Moreover,* $U^p(\sigma) = -L(\mathbf{U}, \mathbf{V}, \mathbf{W})$*,* $\nabla_{\mathbf{M}} L(\sigma)$*, where* $\mathbf{M} \in \{\mathbf{U}, \mathbf{V}, \mathbf{W}\}$*, and given some protagonist at* $\rho \in P$*, if we hold all other agents' strategies fixed,* $U^p(\sigma)$ *is an affine function of the strategy of the protagonist at* ρ*.*

Lemma 0.12. *Assume that for all* $v \in x \cup h$ *and* $v' \in o$*,* f_v *and* $g_{v'}$ *are convex and differentiable, the loss* l *is convex and partially differentiable. Given a joint action* $\mathbf{a} = (\mathbf{U}, \mathbf{V}, \mathbf{W}, \{\mathbf{a}^{(t)}, b^{(t)}\}, \{q_v^{(t)}, d_v^{(t)}\})$ *where all zannis are reasonable, and the antagonist is reasonable, then if the joint action* $(\mathbf{U}, \mathbf{V}, \mathbf{W})$ *for the protagonists is a global minimum , then the protagonists actions are a best response to* $\mathbf{a}$*, and* $\mathbf{a}$ *is a Nash equilibrium.*

Theorem 0.13. *Assume that for all* $v \in x \cup h$ *and* $v' \in o$*,* f_v *and* $g_{v'}$ *are convex and differentiable, the loss* l *is convex and partially differentiable. For every global minimum* $(\mathbf{U}, \mathbf{V}, \mathbf{W})$*, there is a Nash equilibrium where the joint action of the protagonists is* $(\mathbf{U}, \mathbf{V}, \mathbf{W})$*, and for every Nash equilibrium where the joint action of the protagonists is* $(\mathbf{U}, \mathbf{V}, \mathbf{W})$*,* $(\mathbf{U}, \mathbf{V}, \mathbf{W})$ *is a global minimum.*

BIBLIOGRAPHY

BIBLIOGRAPHY

[ABD17] Grigory Antipov, Moez Baccouche, and Jean-Luc Dugelay, *Face aging with conditional generative adversarial networks*, 2017 IEEE International Conference on Image Processing (ICIP), IEEE, 2017, pp. 2089–2093.

[AG18] Marc'Aurelio Ranzato Alex Graves, *Unsupervised Deep Learning Tutorial*, `https://nips.cc/Conferences/2018/Schedule?showEvent=10985/`, 2018, [Online; accessed Decembers-3-2018].

[AGAV09] Enrique Amigó, Julio Gonzalo, Javier Artiles, and Felisa Verdejo, *A comparison of extrinsic clustering evaluation metrics based on formal constraints*, Information retrieval **12** (2009), no. 4, 461–486.

[Agg15] Charu C Aggarwal, *Data mining: The textbook*, Springer, 2015.

[AGMF14] Miguel Araujo, Stephan Günnemann, Gonzalo Mateos, and Christos Faloutsos, *Beyond blocks: Hyperbolic community detection*, 2014.

[AGP+16] Miguel Araujo, Stephan Günnemann, Spiros Papadimitriou, Christos Faloutsos, Prithwish Basu, Ananthram Swami, Evangelos E. Papalexakis, and Danai Koutra, *Discovery of "comet" communities in temporal and labeled graphs com2*, KAIS **46** (2016), no. 3, 657–677.

[AKY99] Charles J Alpert, Andrew B Kahng, and So-Zen Yao, *Spectral partitioning with multiple eigenvectors*, Discrete Applied Mathematics **90** (1999), no. 1, 3–26.

[ASW15] Saeed Aghabozorgi, Ali Seyed Shirkhorshidi, and Teh Ying Wah, *Time-series clustering–a decade review*, Information Systems **53** (2015), 16–38.

[ATK15] Leman Akoglu, Hanghang Tong, and Danai Koutra, *Graph based anomaly detection and description: a survey*, Data mining and knowledge discovery **29** (2015), no. 3, 626–688.

[AW10] Charu C Aggarwal and Haixun Wang, *A survey of clustering algorithms for graph data*, Managing and mining graph data, Springer, 2010, pp. 275–301.

[B+14] Dario Bauso et al., *Game theory: Models, numerical methods and applications*, Foundations and Trends® in Systems and Control **1** (2014), no. 4, 379–522.

[Bal87] Dana H Ballard, *Modular learning in neural networks.*, AAAI, 1987, pp. 279–284.

[Bau08] Ulrike Baur, *Low rank solution of data-sparse sylvester equations*, Numer Linear Algebr **15** (2008), no. 9, 837–851.

[Ber06] Pavel Berkhin, *A survey of clustering data mining techniques*, Grouping multidimensional data, Springer, 2006, pp. 25–71.

[BGLL08] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre, *Fast unfolding of communities in large networks*, JSTAT (2008), no. 10.

[BHKL06] Lars Backstrom, Daniel P. Huttenlocher, Jon M. Kleinberg, and Xiangyang Lan, *Group formation in large social networks: membership, growth, and evolution*, 2006.

[Bis06] Christopher M Bishop, *Pattern recognition and machine learning*, springer, 2006.

[BKM+08] Lars Backstrom, Ravi Kumar, Cameron Marlow, Jasmine Novak, and Andrew Tomkins, *Preferential behavior in online groups*, 2008.

[BKNS00] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander, *Lof: identifying density-based local outliers*, ACM sigmod record, vol. 29, ACM, 2000, pp. 93–104.

[BM07] Avrim Blum and Yishay Monsour, *Learning, regret minimization, and equilibria*.

[BS72] Richard H. Bartels and GW Stewart, *Solution of the matrix equation ax+ xb= c [f4]*, CACM **15** (1972), no. 9.

[BS17] Noam Brown and Tuomas Sandholm, *Safe and nested subgame solving for imperfect-information games*, Advances in Neural Information Processing Systems, 2017, pp. 689–699.

[BUWK15] Nurjahan Begum, Liudmila Ulanova, Jun Wang, and Eamonn Keogh, *Accelerating dynamic time warping clustering with a novel admissible pruning strategy*, KDD, ACM, 2015.

[CB10] Varun Chandola and Arindam Banerjee, *Anomaly detection for discrete sequences: A survey*, IEEE transactions on knowledge and data engineering **24** (2010), no. 5, 823–839.

[CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar, *Anomaly detection: A survey*, ACM computing surveys (CSUR) **41** (2009), no. 3, 15.

[CBLH12] Jeffrey Chan, James Bailey, Christopher Leckie, and Michael Houle, *ciforager: Incrementally discovering regions of correlated change in evolving graphs*, ACM TKDD **6** (2012), no. 3, 11.

[CH94] Diane J. Cook and Lawrence B. Holder, *Substructure Discovery Using Minimum Description Length and Background Knowledge*.

[CKL+09] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, Michael Mitzenmacher, Alessandro Panconesi, and Prabhakar Raghavan, *On Compressing Social Networks*, 2009.

[CLX16] Shaosheng Cao, Wei Lu, and Qiongkai Xu, *Deep neural networks for learning graph representations*, Thirtieth AAAI Conference on Artificial Intelligence, 2016.

[CN11] Adam Coates and Andrew Y Ng, *The importance of encoding versus training with sparse coding and vector quantization*, Proceedings of the 28th international conference on machine learning (ICML-11), 2011, pp. 921–928.

[Cor88] Florence Corpet, *Multiple sequence alignment with hierarchical clustering*, Nucleic acids research **16** (1988), no. 22, 10881–10890.

[CP08] Marcella Corduas and Domenico Piccolo, *Time series clustering and classification by the autoregressive metric*, CSDA **52** (2008), no. 4, 1860–1872.

[CPMF04] Deepayan Chakrabarti, Spiros Papadimitriou, Dharmendra S. Modha, and Christos Faloutsos, *Fully automatic cross-associations*, 2004.

[CS14] Girish Chandrashekar and Ferat Sahin, *A survey on feature selection methods*, Computers & Electrical Engineering **40** (2014), no. 1, 16–28.

[CT12] Thomas M Cover and Joy A Thomas, *Elements of information theory*, John Wiley & Sons, 2012.

[CTF+15] Yongjie Cai, Hanghang Tong, Wei Fan, Ping Ji, and Qing He, *Facets: Fast comprehensive mining of coevolving high-order time series*, KDD, ACM, 2015, pp. 79–88.

[CV05] Rudi Cilibrasi and Paul Vitányi, *Clustering by Compression*, no. 4.

[CVMG+14] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio, *Learning phrase representations using rnn encoder-decoder for statistical machine translation*, arXiv preprint arXiv:1406.1078 (2014).

[CVSFG16] Yoon-Sik Cho, Greg Ver Steeg, Emilio Ferrara, and Aram Galstyan, *Latent space model for multi-modal social data*, WWW, 2016.

[cyt] *clusterMaker: Creating and Visualizing Cytoscape Clusters*, `http://www.cgl.ucsf.edu/cytoscape/cluster/clusterMaker.shtml`.

[Dah96] Rainer Dahlhaus, *On the kullback-leibler information divergence of locally stationary processes*, Stochastic processes and their applications **62** (1996), no. 1.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, arXiv preprint arXiv:1810.04805 (2018).

[DDT+16] Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song, *Recurrent marked temporal point processes: Embedding event history to vector*, KDD, ACM, 2016.

[Dee18] DeepAI, *Build with AI — DeepAI*, `https://deepai.org/machine-learning-glossary-and-terms/unsupervised-learning/`, 2018, [Online; accessed September-30-2018].

[DGK05] Inderjit Dhillon, Yuqiang Guan, and Brian Kulis, *A Fast Kernel-based Multilevel Algorithm for Graph Clustering*, ACM, 2005, pp. 629–634.

[Dhi01] Inderjit S Dhillon, *Co-clustering documents and words using bipartite spectral graph partitioning*, Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2001, pp. 269–274.

[DMBM15] Wim De Mulder, Steven Bethard, and Marie-Francine Moens, *A survey on the application of recurrent neural networks to statistical language modeling*, Computer Speech & Language **30** (2015), no. 1, 61–98.

[Doe16] Carl Doersch, *Tutorial on variational autoencoders*, arXiv preprint arXiv:1606.05908 (2016).

[DPK+17] Pravallika Devineni, Evangelos E. Papalexakis, Danai Koutra, A. Seza Doğruöz, and Michalis Faloutsos, *One size does not fit all: Profiling personalized time-evolving user behaviors*, IEEE/ACM ASONAM, ACM, 2017, pp. 331–340.

[DRS09] Avinash Dixit, David Reiley, and Susan Skeath, *Games of strategies*, 2009.

[DS13] Cody Dunne and Ben Shneiderman, *Motif Simplification: Improving Network Visualization Readability with Fan, Connector, and Clique Glyphs*, Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI), ACM, 2013, pp. 3247–3256.

[DZHL18] Claire Donnat, Marinka Zitnik, David Hallac, and Jure Leskovec, *Learning structural node embeddings via diffusion wavelets*, Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM, 2018, pp. 1320–1329.

[EBC+10] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio, *Why does unsupervised pre-training help deep learning?*, Journal of Machine Learning Research **11** (2010), no. Feb, 625–660.

[EKS+96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al., *A density-based algorithm for discovering clusters in large spatial databases with noise.*, Kdd, vol. 96, 1996, pp. 226–231.

[FCAL16] Chelsea Finn, Paul Christiano, Pieter Abbeel, and Sergey Levine, *A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models*, arXiv preprint arXiv:1611.03852 (2016).

[FFF99] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos, *On Power-law Relationships of the Internet Topology.*

[FM07] Christos Faloutsos and Vasilis Megalooikonomou, *On Data Mining, Compression and Kolmogorov Complexity.*, vol. 15, Springer-Verlag, 2007.

[For10] Santo Fortunato, *Community detection in graphs*, Physics reports **486** (2010), no. 3-5, 75–174.

[FZ16] Leonardo N Ferreira and Liang Zhao, *Time series clustering via community detection in networks*, Information Sciences **326** (2016), 227–242.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep learning*, MIT press, 2016.

[GCB04] Nizar Grira, Michel Crucianu, and Nozha Boujemaa, *Unsupervised and semi-supervised clustering: a brief survey*, A review of machine learning techniques for processing multimedia content **1** (2004), 9–16.

[GERD13] Sean Gilpin, Tina Eliassi-Rad, and Ian Davidson, *Guided learning for role discovery (glrd): Framework, algorithms, and applications*, Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (New York, NY, USA), KDD '13, ACM, 2013, pp. 113–121.

[GH12] Michael U Gutmann and Aapo Hyvärinen, *Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics*, Journal of Machine Learning Research **13** (2012), no. Feb, 307–361.

[GHLR01] Cyril Goutte, Lars Kai Hansen, Matthew G Liptrot, and Egill Rostrup, *Feature-space clustering for fmri meta-analysis*, Hum Brain Mapp **13** (2001), no. 3, 165–183.

[GKSL17] Oshini Goonetilleke, Danai Koutra, Timos Sellis, and Kewen Liao, *Edge labeling schemes for graph data*, ACM, 2017, pp. 12:1–12:12.

[GL16] Aditya Grover and Jure Leskovec, *node2vec: Scalable feature learning for networks*, Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2016, pp. 855–864.

[GMH13] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton, *Speech recognition with deep recurrent neural networks*, 2013 IEEE international conference on acoustics, speech and signal processing, IEEE, 2013, pp. 6645–6649.

[GN02] Michelle Girvan and M. E. J. Newman, *Community structure in social and biological networks*, PNAS **99** (2002).

[GNVL79] Gene Golub, Stephen Nash, and Charles Van Loan, *A hessenberg-schur method for the problem ax+ xb= c*, IEEE Trans Automat Contr. **24** (1979), no. 6.

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, *Generative adversarial nets*, Advances in neural information processing systems, 2014, pp. 2672–2680.

[GTV11] Christos Giatsidis, Dimitrios M Thilikos, and Michalis Vazirgiannis, *Evaluating cooperation in communities with the k-core structure*, IEEE/ACM, 2011.

[HBC+92] Lawrence O Hall, Amine M Bensaid, Laurence P Clarke, Robert P Velthuizen, Martin S Silbiger, and James C Bezdek, *A comparison of neural network and fuzzy clustering techniques in segmenting magnetic resonance images of the brain*, IEEE transactions on neural networks **3** (1992), no. 5, 672–682.

[Hes04] Joao P Hespanha, *An efficient matlab algorithm for graph partitioning*, UCSB, CA, USA (2004).

[HGER+12] Keith Henderson, Brian Gallagher, Tina Eliassi-Rad, Hanghang Tong, Sugato Basu, Leman Akoglu, Danai Koutra, Christos Faloutsos, and Lei Li, *RolX: Structural Role Extraction & Mining in Large Graphs*, ACM, 2012.

[HGL+11] Keith Henderson, Brian Gallagher, Lei Li, Leman Akoglu, Tina Eliassi-Rad, Hanghang Tong, and Christos Faloutsos, *It's who you know: graph mining using recursive structural features.*, KDD, ACM, 2011.

[HKBG08] Christian Hübler, Hans-Peter Kriegel, Karsten Borgwardt, and Zoubin Ghahramani, *Metropolis Algorithms for Representative Subgraph Sampling*, Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (Washington, DC, USA), ICDM '08, IEEE Computer Society, 2008, pp. 283–292.

[HSSK18] Mark Heimann, Haoming Shen, Tara Safavi, and Danai Koutra, *Regal: Representation learning-based graph alignment*, Proceedings of the 27th ACM International Conference on Information and Knowledge Management, ACM, 2018, pp. 117–126.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, *Deep residual learning for image recognition*, Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[Jai10] Anil K Jain, *Data clustering: 50 years beyond k-means*, Pattern Recognit Lett **31** (2010), no. 8, 651–666.

[JHRK19] Di Jin, Mark Heimann, Ryan Rossi, and Danai Koutra, *node2bits: Compact time-and attribute-aware node representations for user stitching*, arXiv preprint arXiv:1904.08572 (2019).

[JK17a] Di Jin and Danai Koutra, *Exploratory analysis of graph data by leveraging domain knowledge*, ICDM17, 2017, pp. 187–196.

[JK17b] Lisa Jin and Danai Koutra, *Ecoviz: Comparative vizualization of time-evolving network summaries*, ACM Knowledge Discovery and Data Mining (KDD) 2017 Workshop on Interactive Data Exploration and Analytics, 2017.

[JLS+17] Di Jin, Aristotelis Leventidis, Haoming Shen, Ruowang Zhang, Junyue Wu, and Danai Koutra, *PERSEUS-HUB: Interactive and Collective Exploration of Large-scale Graphs*, Informatics (Special Issue "Scalable Interactive Visualization") **4** (2017), no. 3.

[Joh67] Stephen C Johnson, *Hierarchical clustering schemes*, Psychometrika **32** (1967), no. 3, 241–254.

[JQV07] Steffen Jorgensen, Marc Quincampoix, and Thomas L Vincent, *Advances in dynamic game theory: Numerical methods, algorithms, and applications to ecology and economics*, vol. 9, Springer Science & Business Media, 2007.

[JRK+18] Di Jin, Ryan Rossi, Danai Koutra, Eunyee Koh, Sungchul Kim, and Anup Rao, *Bridging network embedding and graph summarization*, arXiv preprint arXiv:1811.04461 (2018).

[JRSD15] Abhay Jha, Shubhankar Ray, Brian Seaman, and Inderjit S Dhillon, *Clustering to forecast sparse time-series data*, ICDE, IEEE, 2015.

[KBK00] Stefan Kins, Heinrich Betz, and Joachim Kirsch, *Collybistin, a newly identified brain-specific gef, induces submembrane clustering of gephyrin*, Nature neuroscience **3** (2000), no. 1, 22.

[KF11] U. Kang and Christos Faloutsos, *Beyond 'Caveman Communities': Hubs and Spokes for Graph Compression and Mining*, 2011.

[KF17] Danai Koutra and Christos Faloutsos, *Individual and collective graph mining: Principles, algorithms, and applications*, Synthesis Lectures on Data Mining and Knowledge Discovery **9** (2017), no. 2, 1–206.

[KGP01] Konstantinos Kalpakis, Dhiral Gada, and Vasundhara Puttagunta, *Distance measures for effective clustering of arima time-series*, ICDM, IEEE, 2001.

[KH92] Anders Krogh and John A Hertz, *A simple weight decay can improve generalization*, Advances in neural information processing systems, 1992, pp. 950–957.

[KK99] George Karypis and Vipin Kumar, *Multilevel k-way Hypergraph Partitioning*, 1999.

[KKK+11] Danai Koutra, Tai-You Ke, U Kang, Duen Horng Chau, Hsing-Kuo Kenneth Pao, and Christos Faloutsos, *Unifying Guilt-by-Association Approaches: Theorems and Fast Algorithms*, 2011.

[KKPF13] Danai Koutra, Vasileios Koutras, B. Aditya Prakash, and Christos Faloutsos, *Patterns amongst Competing Task Frequencies: Super-Linearities, and the Almond-DG Model*, 2013.

[KKVF14] Danai Koutra, U Kang, Jilles Vreeken, and Christos Faloutsos, *VoG: Summarizing and Understanding Large Graphs*, 2014.

[kle99] *The Web as a Graph: Measurements, Models and Methods*, 1999.

[KLSH04] Yufeng Kou, Chang-Tien Lu, Sirirat Sirwongwattana, and Yo-Ping Huang, *Survey of fraud detection techniques*, IEEE International Conference on Networking, Sensing and Control, 2004, vol. 2, IEEE, 2004, pp. 749–754.

[KMNR09] Slava Kisilevich, Florian Mansmann, Mirco Nanni, and Salvatore Rinzivillo, *Spatio-temporal clustering*, Data mining and knowledge discovery handbook, Springer, 2009, pp. 855–874.

[KP98] Eamonn J Keogh and Michael J Pazzani, *An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback.*, KDD, vol. 98, 1998.

[KvdOS+17] Nal Kalchbrenner, Aäron van den Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu, *Video pixel networks*, Proceedings of the 34th International Conference on Machine Learning-Volume 70, JMLR. org, 2017, pp. 1771–1779.

[LAL+16] Stefano Leonardi, Aris Anagnostopoulos, Jakub Lacki, Silvio Lattanzi, and Mohammad Mahdian, *Community detection on evolving graphs*, NIPS, 2016.

[LBG+12] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M. Hellerstein, *Distributed graphlab: A framework for machine learning and*

data mining in the cloud, Proceedings of the VLDB Endowment **5** (2012), no. 8, 716–727.

[LCK+10] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani, *Kronecker graphs: An approach to modeling networks*, JMLR **11** (2010), no. Feb, 985–1042.

[LHCG19] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao, *Multi-task deep neural networks for natural language understanding*, arXiv preprint arXiv:1901.11504 (2019).

[Lia05] T Warren Liao, *Clustering of time series dataa survey*, Pattern recognition **38** (2005), no. 11, 1857–1874.

[LK14] Jure Leskovec and Andrej Krevl, *SNAP Datasets: Stanford large network dataset collection*, `http://snap.stanford.edu/data`, June 2014.

[LKF05] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos, *Graphs over time: densification laws, shrinking diameters and possible explanations*, ACM, 2005.

[LLM10] Jure Leskovec, Kevin J Lang, and Michael Mahoney, *Empirical comparison of algorithms for network community detection*, ACM, 2010.

[Llo82] Stuart Lloyd, *Least squares quantization in pcm*, ITIT **28** (1982), no. 2.

[LRU14] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman, *Mining of massive datasets*, Cambridge University Press, 2014.

[LS03] Jennifer M Lee and Erik LL Sonnhammer, *Genomic gene clustering analysis of pathways in eukaryotes*, Genome research **13** (2003), no. 5, 875–882.

[LSDK18] Yike Liu, Tara Safavi, Abhilash Dighe, and Danai Koutra, *Graph summarization methods and applications: A survey*, ACM CSUR **51** (2018), no. 3, 62.

[LSK15] Yike Liu, Neil Shah, and Danai Koutra, *An empirical comparison of the summarization power of graph clustering methods*, Neural Information Processing Systems (NIPS) Networks Workshop, Montreal, Canada (2015).

[LSSK18] Yike Liu, Tara Safavi, Neil Shah, and Danai Koutra, *Reducing large graphs to small supergraphs: a unified approach*, Social Network Analysis and Mining **8** (2018), no. 1, 17.

[LT10] Kristen LeFevre and Evimaria Terzi, *Grass: Graph structure summarization.*, SIAM, 2010.

[LZS+19] Yike Liu, Linhong Zhu, Pedro Szekely, Aram Galstyan, and Danai Koutra, *Coupled clustering of time-series and networks.*

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean, *Efficient estimation of word representations in vector space*, arXiv preprint arXiv:1301.3781 (2013).

[MH08] Laurens van der Maaten and Geoffrey Hinton, *Visualizing data using t-sne*, Journal of machine learning research **9** (2008), no. Nov, 2579–2605.

[MKB+10] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur, *Recurrent neural network based language model*, Eleventh annual conference of the international speech communication association, 2010.

[MLKCW03] Carla S Möller-Levet, Frank Klawonn, Kwang-Hyun Cho, and Olaf Wolkenhauer, *Fuzzy clustering of short time-series and unevenly distributed sampling points*, IDA, Springer, 2003.

[MLR19] Geoffrey J McLachlan, Sharon X Lee, and Suren I Rathnayake, *Finite mixture models*, Annual review of statistics and its application **6** (2019), 355–378.

[NG04] Mark E. J. Newman and Michelle Girvan, *Finding and Evaluating Community Structure in Networks*, Physical Review E **69** (2004), no. 2, 026113+.

[NRS08] Saket Navlakha, Rajeev Rastogi, and Nisheeth Shrivastava, *Graph Summarization with Bounded Error*, 2008.

[OCP14] OCP, *Open Connectome Project*, http://www.openconnectomeproject.org, 2014.

[OFC99] Tim Oates, Laura Firoiu, and Paul R Cohen, *Clustering time series with hidden markov models and dynamic time warping*, IJCAI Workshop, 1999.

[OKK16] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu, *Pixel recurrent neural networks*, arXiv preprint arXiv:1601.06759 (2016).

[OLV18] Aaron van den Oord, Yazhe Li, and Oriol Vinyals, *Representation learning with contrastive predictive coding*, arXiv preprint arXiv:1807.03748 (2018).

[PARS14] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena, *Deepwalk: Online learning of social representations*, Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2014, pp. 701–710.

[PG15] John Paparrizos and Luis Gravano, *k-shape: Efficient and accurate clustering of time series*, SIGMOD, 2015.

[PMB13] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio, *On the difficulty of training recurrent neural networks*, International conference on machine learning, 2013, pp. 1310–1318.

[PNI+18] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer, *Deep contextualized word representations*, arXiv preprint arXiv:1802.05365 (2018).

[PPK+14] Manish Purohit, B Aditya Prakash, Chanhyun Kang, Yao Zhang, and VS Subrahmanian, *Fast influence-based coarsening for large networks*, ACM, 2014, pp. 1296–1305.

[PS83] Girish Punj and David W Stewart, *Cluster analysis in marketing research: Review and suggestions for application*, Journal of marketing research **20** (1983), no. 2, 134–148.

[PS89] David Peleg and Alejandro A. Schäffer, *Graph spanners*, Journal of Graph Theory **13** (1989), no. 1, 99–116.

[PSS+10] B. Aditya Prakash, Mukund Seshadri, Ashwin Sridharan, Sridhar Machiraju, and Christos Faloutsos, *EigenSpokes: Surprising Patterns and Scalable Community Chipping in Large Graphs*.

[PY10] Sinno Jialin Pan and Qiang Yang, *A survey on transfer learning*, IEEE Transactions on knowledge and data engineering **22** (2010), no. 10, 1345–1359.

[Ran71] William M Rand, *Objective criteria for the evaluation of clustering methods*, Journal of the American Statistical association **66** (1971), no. 336, 846–850.

[RGM03] Sriram Raghavan and Hector Garcia-Molina, *Representing web graphs*, IEEE, 2003.

[RGSB14] Matteo Riondato, David García-Soriano, and Francesco Bonchi, *Graph summarization with quality guarantees*, IEEE, 2014, pp. 947–952.

[RHW85] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams, *Learning internal representations by error propagation*, Tech. report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[Ris83] Jorma Rissanen, *A Universal Prior for Integers and Estimation by Minimum Description Length*, no. 2.

[RNSS18] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever, *Improving language understanding by generative pre-training*, URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/languageunsupervised/language understanding paper. pdf (2018).

[RSF17] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo, *struc2vec: Learning node representations from structural identity*, Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2017, pp. 385–394.

[Sch07] Satu Elisa Schaeffer, *Graph clustering*, Computer science review **1** (2007), no. 1, 27–64.

[SEB+18] Othman Sbai, Mohamed Elhoseiny, Antoine Bordes, Yann LeCun, and Camille Couprie, *Design: Design inspiration from generative networks*, Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 0–0.

[SFH17] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton, *Dynamic routing between capsules*, Advances in neural information processing systems, 2017, pp. 3856–3866.

[SHB15] Rico Sennrich, Barry Haddow, and Alexandra Birch, *Neural machine translation of rare words with subword units*, arXiv preprint arXiv:1508.07909 (2015).

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, *Dropout: a simple way to prevent neural networks from overfitting*, The Journal of Machine Learning Research **15** (2014), no. 1, 1929–1958.

[SK92] CT Shaw and GP King, *Using cluster analysis to classify time series*, Physica D: Nonlinear Phenomena **58** (1992), no. 1-4, 288–298.

[SKZ+15] Neil Shah, Danai Koutra, Tianmin Zou, Brian Gallagher, and Christos Faloutsos, *Timecrunch: Interpretable dynamic graph summarization*, ACM, 2015.

[SP97] Mike Schuster and Kuldip K Paliwal, *Bidirectional recurrent neural networks*, IEEE Transactions on Signal Processing **45** (1997), no. 11, 2673–2681.

[SS11] Daniel A. Spielman and Nikhil Srivastava, *Graph sparsification by effective resistances*, SIAM J. Comput. **40** (2011), no. 6, 1913–1926.

[SSK18] Tara Safavi, Chandra Sripada, and Danai Koutra, *Fast network discovery on sequence data via time-aware hashing*, Knowledge and Information Systems (2018), 1–31.

[SV09] Thomas Strohmer and Roman Vershynin, *A randomized kaczmarz algorithm with exponential convergence*, JFAA **15** (2009), no. 2.

[SZ16] Dale Schuurmans and Martin A Zinkevich, *Deep learning games*, Advances in Neural Information Processing Systems, 2016, pp. 1678–1686.

[TQW+15] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei, *Line: Large-scale information network embedding*, Proceedings of the 24th international conference on world wide web, International World Wide Web Conferences Steering Committee, 2015, pp. 1067–1077.

[TZHH11] Hannu Toivonen, Fang Zhou, Aleksi Hartikainen, and Atte Hinkka, *Compression of Weighted Graphs*, 2011, pp. 965–973.

[VDODZ+16] Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu, *Wavenet: A generative model for raw audio.*, SSW **125** (2016).

[VPT16] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba, *Generating videos with scene dynamics*, Advances In Neural Information Processing Systems, 2016, pp. 613–621.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin, *Attention is all you need*, Advances in neural information processing systems, 2017, pp. 5998–6008.

[WFY18] Zheng Wang, Kun Fu, and Jieping Ye, *Learning to estimate the travel time*, Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM, 2018, pp. 858–866.

[XW05] Rui Xu and Donald C Wunsch, *Survey of clustering algorithms*.

[XY04] Yimin Xiong and Dit-Yan Yeung, *Time series clustering with arma mixtures*, Pattern Recognition **37** (2004), no. 8, 1675–1689.

[YL13] Jaewon Yang and Jure Leskovec, *Overlapping community detection at scale: a nonnegative matrix factorization approach*, ACM, 2013.

[YZD+19] Yujun Yan, Jiong Zhu, Marlena Duda, Eric Solarz, Chandra Sripada, and Danai Koutra, *Groupinn: Grouping-based interpretable neural network for classification of limited, noisy brain data*.

[ZF18] Arthur Zimek and Peter Filzmoser, *There and back again: Outlier detection between statistical reasoning and data mining algorithms*, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **8** (2018), no. 6, e1280.

[ZGGS+16] Linhong Zhu, Majid Ghasemi-Gol, Pedro Szekely, Aram Galstyan, and Craig A Knoblock, *Unsupervised entity resolution on multi-type graphs*, International Semantic Web Conference, Springer, 2016, pp. 649–667.

[Zhu05] Xiaojin Jerry Zhu, *Semi-supervised learning literature survey*, Tech. report, University of Wisconsin-Madison Department of Computer Sciences, 2005.

[ZJBP08] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione, *Regret minimization in games with incomplete information*, Advances in neural information processing systems, 2008, pp. 1729–1736.

[ZLPS17] Manzil Zaheer, Chun-Liang Li, Barnabás Póczos, and Ruslan Salakhutdinov, *Gan connoisseur: Can gans learn simple 1d parametric distributions?*

[ZSV14] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals, *Recurrent neural network regularization*, arXiv preprint arXiv:1409.2329 (2014).

[ZSY+15] Liang Zhao, Qian Sun, Jieping Ye, Feng Chen, Chang-Tien Lu, and Naren Ramakrishnan, *Multi-task learning for spatio-temporal event forecasting*, KDD, ACM, 2015.

[ZZX08] Zhe Zhang, Junxi Zhang, and Huifeng Xue, *Improved k-means clustering algorithm*, 2008 Congress on Image and Signal Processing, vol. 5, IEEE, 2008, pp. 169–172.