University of California
Santa Barbara

Learning with Richly Structured Data

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Computer Science

by

Furkan E. Kocayusufoglu

September 2021

The Dissertation of Furkan E. Kocayusufoglu is approved.

Professor Xifeng Yan

Professor Francesco Bullo

Professor Yu-Xiang Wang

Professor Ambuj Singh, Committee Chair

September 2021

Learning with Richly Structured Data

To my family for their support, inspiration and love.

Acknowledgements

I would like to express my heartfelt gratitude to my research advisor Ambuj K. Singh for being a great mentor, supporting my research and ideas, and allowing me to learn and grow so much as part of his group. Thanks to him, I am completing my PhD studies with many valuable skills and a vision for research, both of which will benefit me greatly during my career. Thanks to Xifeng Yan, Francesco Bullo, and Yu-Xiang Wang for being part of my committee, being generous with their expertise and time, and providing valuable mentorship. Thanks also to Hakan Ferhatosmanoglu and Ozcan Ozturk for inspiring and motivating me to apply for a PhD during my undergraduate studies.

I feel fortunate to have collaborated with many brilliant researchers: Arlei Silva, Hongyuan You, Minh Hoang, Saber Jafarpour, Sandeep Tata, Ying Sheng, Tao Wu, Heng-Tze Cheng. I also would like to thank my labmates—Sourav, Haraldur, Omid, Alex, Rachel, Sikun, Mert, Zexi, and Saurabh—for contributing to my journey both as a researcher and as a human. My special thanks go to Arlei Silva and Sourav Medya for helping me navigate this long and challenging journey. They both have been great mentors and friends from the start of my PhD studies. I am also grateful to Tim Robinson from the UCSB CS staff for his help and support.

Being far from my family, I was extremely lucky to have found another Turkish family in Santa Barbara. I would like to thank Arda, Faruk, Semih, Bugra, Asutay, Berkay, Metehan, Can, Mert, and Arinc for the great memories and their lifelong friendships.

Finally, I can not thank my family enough for their constant love and unconditional support in pursuing my goals, even if it meant being away from their beloved son for many years. Thanks to my father Dr. Ismail Kocayusufoglu for introducing me to Mathematics at a young age, and my mother Aysun Kocayusufoglu for introducing me to all other aspects of life that can not (yet) be explained in a mathematical form.

The research developed during my PhD studies has been generously supported by the National Science Foundation and the Defense Threat Reduction Agency.

Curriculum Vitæ

Furkan E. Kocayusufoglu

Education

2016-2021	PhD in Computer Science, University of California, Santa Barbara.
2011-2016	BSc in Computer Science, Bilkent University, Ankara, Turkey.

Experience

09/2018-09/2021	*Graduate Student Researcher*, UC Santa Barbara, CA.
06/2020-09/2020	*Research Intern*, Google Brain, Mountain View, CA.
06/2019-09/2019	*Software Engineer Intern*, Facebook, New York City, NY.
06/2018-09/2018	*Research Intern*, Google Research, Mountain View, CA.
06/2017-09/2017	*Teaching Assistant*, UC Santa Barbara, CA.
06/2017-09/2017	*Research Intern*, Toyon Research Corporation, Santa Barbara, CA.
09/2016-06/2018	*IGERT Fellow*, UC Santa Barbara, CA.

Publications

Refereed Journals and Conferences:

Arlei Silva, Furkan Kocayusufoglu, Saber Jafarpour, Francesco Bullo, Ananthram Swami, Ambuj Singh. *Combining Physics and Machine Learning for Network Flow Estimation.* International Conference on Learning Representations (ICLR), 2020.

Hongyuan You, Furkan Kocayusufoglu, Ambuj Singh. *DANR: Discrepancy-aware Network Regularization.* SIAM International Conference on Data Mining (SDM), 2020.

Furkan Kocayusufoglu, Ying Sheng, Nguyen Vo, James Wendt, Qi Zhao, Sandeep Tata, Marc Najork. *Riser: Learning better representations for richly structured emails.* The Web Conference (WWW), 2019.

Furkan Kocayusufoglu, Minh X Hoang, Ambuj Singh. *Summarizing network processes with network-constrained Boolean matrix factorization.* IEEE International Conference on Data Mining (ICDM), 2018.

Elif Eser, Furkan Kocayusufoglu, Bahaeddin Eravci, Hakan Ferhatosmanoğlu, Josep L Larriba-Pey. *Generating time-varying road network data using sparse trajectories.* IEEE 16th International Conference on Data Mining Workshops (ICDMW), 2016.

Under Review:

Furkan Kocayusufoglu, Arlei Silva, Ambuj Singh. *FlowGEN: A Generative Model for Flow Graphs*, 2021.

Furkan Kocayusufoglu, Tao Wu, Anima Singh, Georgios Roumpos, Heng-Tze Cheng, Sagar Jain, Ed Chi, Ambuj Singh. *Multi-Resolution Attention for Personalized Item Search*, 2021.

Working Papers:

Furkan Kocayusufoglu, Arlei Silva, Francesco Bullo, Ananthram Swami, Ambuj Singh. *Improving Network Flow Estimation via Knowledge Transfer*, 2021.

Abstract

Learning with Richly Structured Data

by

Furkan E. Kocayusufoglu

This thesis is dedicated to learning structured representations of data, to be utilized by a diverse set of downstream tasks emerging in numerous application domains such as user modeling, document categorization, and graph representation learning. Despite their notable contextual differences, the common objective behind these studies is our desire to incorporate the rich structure of data into our learned representations (both discrete and continuous) and computations, in ways that are unique to each problem. We provide comprehensive evidence showing that the principled utilization of data structure, regardless of the problem domain, is a key step towards reaching our learning objectives.

The second half of the thesis specifically targets problems centered around a complex family of graphs called flow graphs. Besides nodes and edges, flow graphs capture edge flows of a quantity of interest (*e.g.*, water, power, people) being transported through the graph. As these flows often possess higher-order graph-level dynamics driven by sources/destinations, hotspots, and domain-specific physics, the underlying structure of flow graphs poses unique challenges from a learning viewpoint. In particular, we study two challenging problems: (i) network flow estimation based on partial flow observations, and (ii) data-driven generation of realistic flow graphs as an alternative to domain-specific simulations. During each of these studies, we showcase the complex structural patterns emerging in many real-world flow graphs and propose novel methodologies that can account for such richly structured information.

Permissions and Attributions

This dissertation contains material that has been published or is in the process of being published. The author of this dissertation claims major contributions in the development of the research works described below:

1. The content of Chapter 2 has been previously published as: Furkan Kocayusufoglu, Minh X Hoang, Ambuj Singh. *Summarizing network processes with network-constrained Boolean matrix factorization.* IEEE International Conference on Data Mining (ICDM), 2018. DOI: 10.1109/ICDM.2018.00039.

2. The content of Chapter 3 has been previously published as: Furkan Kocayusufoglu, Ying Sheng, Nguyen Vo, James Wendt, Qi Zhao, Sandeep Tata, Marc Najork. *Riser: Learning better representations for richly structured emails.* The Web Conference (WWW), 2019. DOI: 10.1145/3308558.3313720.

3. The content of Chapter 6.1 has been previously published as: Hongyuan You, Furkan Kocayusufoglu, Ambuj Singh. *DANR: Discrepancy-aware Network Regularization.* SIAM International Conference on Data Mining (SDM), 2020. DOI: 10.1137/1.9781611976236.24.

For ACM, authors can include partial or complete papers of their own in a dissertation as long as citations and DOI pointers to the Versions of Record in the ACM Digital Library are included. If interested in reprinting or republishing ACM copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please contact the ACM.

For IEEE, requirements to be followed when using an entire IEEE copyrighted paper in a thesis: 1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]; 2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line; 3) In placing the thesis on the author's university website, please display the following message

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A common characteristic of the problems investigated in this thesis is that they are driven by real applications and the availability of real data. Learning problems in the real world often involve complex and richly structured data, coupled with a diverse set of learning objectives depending on the application domain. Despite the differences across many domains, the underlying data structure inevitably shapes how the learning algorithms are derived for satisfactory results on the downstream task of interest.

For instance, how can we leverage the underlying templatic HTML structure of emails to help us learn better representations of emails for classification tasks? Or, how can we utilize the connectivity patterns of roads in order to make more accurate traffic forecasts in the case of a traffic accident or road closure? The shared challenge behind these problems (and many others) is, *how can we construct principled methods that successfully account for the underlying data structure?* The goal is to design methods with relational inductive biases [1], based on our understanding of the data and its structure, that can generalize a finite set of observations into a general model of the domain. An ability to model and manipulate structured knowledge allows learning structured representations, which in turn enables an essential ingredient for human-like intelligence, that is, structured behaviors.

A significant part of this dissertation focuses on *graph* (or *network*) structured data, consequently, on learning problems on graphs. Graphs are a powerful theoretical framework for modeling complex data, and they naturally appear in numerous domains including social sciences, physics, biology, chemistry, geography, and computer science, to name a few. Broadly speaking, we study attributed graphs that couple spatially local structures with higher-order (global) constraints (*e.g.* physics) in domain-specific ways. Understanding the interplay between these two modalities is our main research objective. In particular, most of the problems discussed in this dissertation focus on kinds of graph structures that emerge as a result of complex processes within certain geographical regions of different scales. Example problems include housing price prediction within neighborhoods of the city, traffic flow estimation within large road networks, and generation of realistic intra-city human mobility networks of different days and times of the week.

This dissertation also covers studies in which the data emerge in the form of a (1) time-ordered sequence of entities and (2) HTML markup structure. Notice that it is possible to see these structures as a special form of a graph where sequential input is analogous to a line graph, while the HTML markup structure can essentially be represented by a Document Object Model (DOM) tree. Although these structures are historically studied under different problem domains (and by different communities) with seemingly different modeling desiderata, most recent advancements in the literature show traces of evidence that sharing knowledge across these domains can show remarkable results[].

Subject areas: The chapters of this thesis show footprints of a diverse set of paradigms from the fields of machine learning, optimization, and statistics. Namely, these paradigms include geometric learning, user modeling, semi-supervised learning, representation learning, generative modeling, meta-learning, transfer learning, bi-level optimization, distributed optimization, and sampling.

1.1 Contributions and Organization

The contributions of this dissertation can be summarized as follows:

1. *Summarizing network processes with network-constrained Boolean matrix factorization [2] (Chapter 2):* Understanding and modeling complex network processes is an important task in many real-world applications. The first challenge is to discover patterns in such complex data. In this work, our goal is to *summarize different processes in a network by a small yet interpretable set of network patterns, each of which represents a local community of connected nodes frequently participating in the same network processes.* We formulate this problem as a Boolean Matrix Factorization with a network constraint, which we prove to be NP-hard. We then propose an efficient algorithm that incrementally adds the best patterns and achieve scalability with two further improvements. First, to decide which network processes contain which network patterns, we introduce two mapping algorithms with linear costs. Second, to systematically mine the exponential subgraph search space for good patterns, we devise two sampling algorithms based on Monte Carlo Markov Chain. Experimental results on both synthetic and real-world datasets show that our solutions are scalable and find network patterns that effectively summarize network processes.

2. *Riser: Learning better representations for richly structured emails [3] (Chapter 3):* Recent studies show that an overwhelming majority of emails are machine-generated and sent by businesses to consumers. Many large email services are interested in extracting structured data from such emails to enable intelligent assistants. This allows experiences like being able to answer questions such as "What is the address of my hotel in New York?" or "When does my flight leave?". A high-quality email classifier is a critical piece in such a system. In this work, we argue that the rich

formatting used in business-to-consumer emails contains valuable information that can be used to learn better representations. Most existing methods focus only on textual content and ignore the rich HTML structure of emails. We introduce *RiSER* (Richly Structured Email Representation) – an approach for incorporating both the structure and content of emails. RiSER projects the email into a vector representation by jointly encoding the HTML structure and the words in the email. We then use this representation to train a classifier. To our knowledge, this is the first description of a neural technique for combining formatting information along with the content to learn improved representations for richly formatted emails. Experimenting with a large corpus of emails received by users of Gmail, we show that RiSER outperforms strong attention-based LSTM baselines. We expect that these benefits will extend to other corpora with richly formatted documents. We also demonstrate with examples where leveraging HTML structure leads to better predictions.

3. *Multi-Resolution Attention for Personalized Item Search (Chapter 4):* Personalized item search has become an essential tool for online platforms—where users interact with a large corpus of items (*e.g.*, click, purchase, like) via a search query—to provide their users with a more satisfactory search experience. The record (or history) of users' past interactions serves as a valuable asset to achieve personalization. While user history data can span over a long period of time, only a part of the history is relevant to a user's current search intent. Moreover, since historical interactions take place at aperiodic points in time, modeling their relevance to the current search query entangles complex temporal dependencies. We propose *multi-resolution attention* to address these challenges for personalized item search. Our approach captures higher-order temporal relations between user queries and their history

across several temporal subspaces (*i.e.*, resolutions), each corresponding to distinct temporal ranges with *adaptive* time boundaries that are also learned directly from data. We achieve this by coupling the conventional multi-head attention module with a differentiable soft-thresholding mechanism, which essentially operates as a masking function in the temporal domain. Comparisons with strong baselines on an open-source benchmark dataset confirm the efficacy of our approach.

4. *FlowGEN: A Generative Model for Flow Graphs (Chapter 5):* Flow graphs capture the directed flow of a quantity of interest (*e.g.*, water, power, vehicles) being transported through an underlying structure. Modeling and generating realistic flow graphs is key in many applications in infrastructure design, transportation, planning, biomedical and social sciences. However, they pose a great challenge to existing generative models due to their complex dynamics that is often governed by domain-specific physical laws. We introduce FlowGEN, an implicit generative model for flow graphs. FlowGEN learns how to jointly generate graph topologies and flows with diverse dynamics directly from data using a novel (flow) graph neural network. We conduct experiments over a diverse set of real-world and simulated networks, including transportation, power transmission, and water networks, Our results show that the proposed approach is able to effectively reproduce relevant local and global properties of flow graphs, including flow conservation, cyclic trends, and congestion around hotspots.

5. *Problems on Network Regularization (Chapter 6):* Here, we grouped problems that propose regularization techniques for graph-structured data, which can be applied to a wide variety of real-world semi-supervised regression and classification problems over graphs. These problems include but not limited to housing price estimation, water quality estimation of lakes, vehicle (or current) flow estimation of roads (power

transmission lines). Notice that all of these problems can be defined over graphs that are constructed from certain geographical regions (or boundaries) of different scales (*e.g.*, cities, counties, states, countries), coupled with local contextual features.

(a) *DANR: Discrepancy-aware Network Regularization [4] (Section 6.1):* Network regularization is an effective tool for incorporating structural prior knowledge to learn coherent models over networks, and has yielded provably accurate estimates in applications ranging from spatial economics to neuroimaging studies. Recently, there has been an increasing interest in extending network regularization to the spatio-temporal case to accommodate the evolution of networks. However, in both static and spatio-temporal cases, missing or corrupted edge weights can compromise the ability of network regularization to discover desired solutions. To address these gaps, we propose a novel approach—*discrepancy-aware network regularization* (DANR)—that is robust to inadequate regularizations and effectively captures model evolution and structural changes over spatio-temporal networks. We develop a distributed and scalable algorithm based on alternating direction method of multipliers (ADMM) to solve the proposed problem with guaranteed convergence to global optimum solutions. Experimental results on both synthetic and real-world networks demonstrate that our approach achieves improved performance on various tasks, and enables interpretation of model changes in evolving networks.

(b) *Combining Physics and Machine Learning for Network Flow Estimation [5] (Section 6.2):* The flow estimation problem consists of predicting missing edge flows in a network (e.g., traffic, power, and water) based on partial observations. These missing flows depend both on the underlying *physics* (edge features and a flow conservation law) as well as the observed edge flows. This paper introduces an optimization framework for computing missing edge flows and solves the problem using bilevel optimization and deep learning. More specifically, we learn regularizers that depend on edge features (e.g., number of lanes in a road, the resistance of a power line) using neural networks. Empirical results show that our method accurately predicts missing flows, outperforming the best baseline, and is able to capture relevant physical properties in traffic and power networks.

Statement of the thesis:

Building models using structured representations of data (i) allows us to better generalize to the problem domain, and (ii) helps mitigate the effects of data scarcity.

Chapter 2

Network-constrained Boolean Matrix Factorization

2.1 Introduction

Binary datasets appear naturally in many domains. Summarizing, describing, and analyzing the underlying structure of such datasets is the fundamental objective of Boolean Matrix Factorization (BMF). In the context of BMF [6], a binary matrix is composed of observations (rows), each including a set of participants (columns). BMF aims to decompose a binary matrix $S \in \{0,1\}^{m\times n}$ into a pair of low-rank binary matrices, $M \in \{0,1\}^{m\times k}$ and $B \in \{0,1\}^{k\times n}$, where $S \approx M \odot B$ and $\odot$ is the Boolean matrix product. Following this decomposition, the latter matrix B contains "meaningful" patterns and the matrix M maps these patterns to the observations.

Consider a road network, for instance in Fig. 2.1, where nodes represent road segments and edges exist between two segments if they are physically connected. Based on these road networks, GPS route planning applications are collecting millions of car trajectories every day, projected on thousands of road segments. Each trajectory represents a car ride of a user by capturing all the road segments passed through during a ride. One simple way to store these trajectories is to represent them with a binary matrix, where rows represent trajectories, and columns represent road segments. A useful first step in the analysis of this data would be to compress the large amount of raw trajectories into a summarization of the most frequent paths. For example, the set of red road segments in Fig. 2.1 are frequently travelled across by numerous trajectories. Such a frequent set of road segments suggests a busy route to which the city planners need to pay attention.

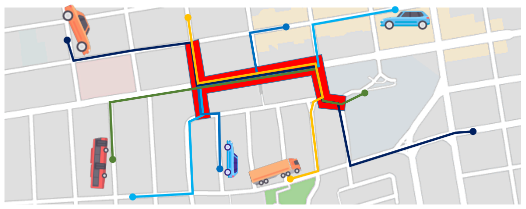


Figure 2.1: Trajectories and frequently used road segments

In the above example, participants (road segments) of a certain observation (car trajectory) have additional underlying constraints: for them to co-occur in an observation, they need to be connected through an underlying network structure. Similar network structures exist in many types of real-world data, guiding various processes generating the data. Examples of such network processes include information spreads in social networks, congestion in traffic networks, the spread of failures in computer networks, and the activation of brain regions in a brain network when a person performs a task. Therefore, mined patterns from these data are of greater interest if they reflect the network structure.

Previous BMF methods have tried to minimize the reconstruction error, without taking into account the aforementioned network constraints. Doing so sometimes yields undesirable patterns that (a) do not correspond to an underlying network structure, and hence can not capture the network effect, and (b) are vastly affected by noise and missing values in data, since classic BMF methods do not leverage the underlying network

Examples	**Social network**	**Computer network**	**Traffic network**	**Brain network**	**Software function call network**
Nodes	Users	Computer servers	Road segments	Brain regions	Functions
Edges	Friendship	Connections	Intersections	Correlations	Call relationship
Network process	Spread of information among users through their social connections.	Spread of failure or overload.	Spread of traffic jam	Activity of brain regions during a task.	Function calls.
Network pattern	A local connected community that frequently reacts to the same content.	A set of connected servers that often fail together.	A set of frequently jammed connected road segments.	A set of brain regions active together during a task.	A frequent function call path.
Use of patterns	Target and control information spread.	Redesign network to avoid the failed patterns.	Redesign the network to reduce traffic bottlenecks.	Model brain activity and relate different tasks.	Optimize frequent paths to reduce latency.

Table 2.1: Five different example applications of network-constrained patterns.

structure that generates the given observations in the first place. As a result, these patterns have less explanatory power, considering they are not capable of summarizing unseen observations or predicting future behaviours.

In this paper, we propose a novel problem, *netBMF: network constrained Boolean Matrix Factorization*. Our goal is to find the top-k network patterns that most effectively summarize complex network processes. We define a network pattern as a *connected* subnetwork to capture the effect of the network structure on the network processes. A pattern is essentially a local community of nodes with similar dynamic behaviors, that strongly and repeatedly correlate in their actions and behaviors in various network processes. Such a pattern looks beyond the set of active nodes and pinpoints the different impacts of different parts of the network on a network process. Identifying these patterns is helpful for understanding [7], controlling [8], and predicting [9] the behaviors of network processes as exemplified in Table 2.1.

Our contributions are as follows:

- We introduce network-constrained Boolean Matrix Factorization (netBMF) as a novel data mining challenge to find the top-k *connected* network patterns that summarize complex network processes. netBMF also reduces the effect of noise by leveraging the underlying network structure of the data.

- We prove that netBMF is NP-hard and then propose a Metropolis-Hastings based sampling algorithm that effectively summarizes the network processes. The proposed solutions are generic, scalable and can be applied to the classic BMF problem when the dataset is huge and has network constraints. We also introduce four variants of our algorithm to enable various sampling strategies.

- Extensive experimental results on both synthetic and four real-world datasets indicate that, discovering explanatory patterns from the data, our proposed algorithms achieve lower reconstruction errors than all other baseline methods, and scale linearly with its parameters. Code and data are made available online [1].

The rest of this work is organized as follows: Section 2.2 formally defines the problem. Section 2.3 discusses the related work and possible baseline approaches. Section 2.4 explains the computational challenges and details the proposed solutions. Section 5.4 presents the experimental results. Finally, Section 5.5 concludes the work.

2.2 The netBMF Problem

Before defining the netBMF problem, we first introduce some preliminary notation. Let $G = (V, E)$ denote an undirected network, where $V = \{v_1, \ldots, v_n\}$ is the set of nodes and E is the set of edges, and let $P = \{p_1, \ldots, p_m\}$ be a set of network processes, where p_i represents some process infecting nodes in G under the guidance of the network structure. The *network state* $s_i \subseteq V$ is then the *final set of participating nodes* in a network process p_i. Using these fundamental concepts, we now define *state matrix* in the context of netBMF, followed by the formal definition of our problem.

Definition 1 (State matrix) *The* state matrix *of a set of network states* $\mathcal{S} = \{s_1, \ldots, s_m\}$

[1] https://github.com/FurkanKyo/netBMF

is $S \in \{0,1\}^{m\times n}$, where $S_{i,j} = 1$ iff $v_j \in s_i$, i.e., node v_j is present in the network state s_i.

Problem 1 (netBMF) *Given an undirected network $G = (V, E)$, a state matrix $S \in \{0,1\}^{m\times n}$, and the number of patterns k, network-constrained Boolean Matrix Factorization (netBMF) finds a pattern matrix $B \in \{0,1\}^{k\times n}$, and a mapping matrix $M \in \{0,1\}^{m\times k}$ s.t.:*

1. *The summarization error $g(S, B, M)$ is minimized,*
   where $g(S, B, M) = ||S \oplus (M \odot B)||_F^2$
2. *Each row of B corresponds to an induced undirected subgraph in G, i.e., $G_q = (V_q, E_q)$ is connected $\forall q = 1, \ldots, k$, where $V_q = \{v_j \in V \mid B_{q,j} = 1\}$, $E_q = \{(v_j, v_{j'}) \in E \mid v_j, v_{j'} \in V_q\}$.*

Here $\odot$ represents the Boolean matrix product [6], while $\oplus$ represents the XOR operation. Since all matrices and operators are Boolean, the squared Frobenius norm effectively counts the number of different bits between S and $M \odot B$. Condition (1) encourages *accurate* summarization of the given binary state matrix. Furthermore, the patterns are *spatially-robust* because both negative and positive errors are allowed. Condition (2) ensures the *connected* characteristic. The connectedness of each pattern guarantees the existence of a path between every pair of nodes in the induced subgraph, which includes all nodes in the pattern, as well as edges with both end nodes in the pattern.

For the remainder of the paper, we refer to binary matrix B as **net-basis** and **pattern matrix** interchangeably. In the context of netBMF, rows of B then corresponds to **patterns** that represent sets of connected nodes (participants) that often co-occur in same processes (observations).

2.3 Related Work

Here we discuss related works from two main categories: frequent pattern mining and Boolean matrix factorization.

Frequent Pattern Mining. Frequent pattern mining (*FPM*) has become a fundamental problem in data mining research and has been studied extensively in various different forms. Here, we examine frequent pattern mining in two main settings: (a) frequent itemset mining, where pattern refers to an itemset, which is a collection of items that frequently co-occur in large number of transactions, and (b) frequent subgraph mining, where pattern refers to a graph pattern, which is a connected (often smaller) subgraph that frequently appears in a large set of other subgraphs.

Regardless of the categories mentioned above, a well known problem in FPM is that the number of frequent patterns is often huge. A large frequent pattern will contain an exponential number of smaller frequent sub-patterns, since each sub-pattern of a frequent pattern is also frequent. Therefore, it has been argued that some patterns are redundant because they have identical support as their super-patterns. In order to overcome this redundancy, different variants of FPM problem are proposed, such as closed frequent pattern mining [10] and maximal frequent pattern mining [11].

a. Frequent Itemset Mining. In [12], a pattern refers to set of items, and therefore the objective is to either find or approximate (depending on how strict the support of an itemset is defined) frequent/closed/maximal itemsets that have frequency more than a given threshold. Frequent itemset mining approaches are somewhat related to our problem as they can be employed to summarize large transitional databases [13]. In [14], the authors show that frequent closed itemsets can be used as good initialization candidates to the boolean matrix decomposition algorithms. However, when it comes to summarization aspects, all these methods suffer from redundancy among frequent itemsets as well as determining the value of support prior to computing the factorization since frequent closed itemsets tend to lose the ability to summarize with increasing support values.

b. Frequent Subgraph Mining. Graph related frequent pattern mining approaches refer to pattern as a connected subgraph, that frequently appears in the large set of other subgraphs. One of the most popular tool for this problem is gSpan [15], which uses exact matching (a pattern must lie completely within a network process), and hence isn't spatially robust. Besides the classic frequent subgraph mining problems, closed and maximal frequent subgraph mining problems are also studied excessively. As an extension to gSpan, the CloseGraph algorithm is proposed to tackle the closed frequent subgraph mining problem [16]. Other pattern-growth based approaches such as SPIN [17] and Margin [18] only mine maximal frequent subgraphs in order to reduce redundancy.

Frequent subgraph mining methods are more related to our problem because they ensure connectivity, condition (2) of the netBMF problem. However, top-k frequent patterns usually correspond to the most active small groups of nodes in the graph. Despite various kinds of enhancements, the closed and maximal frequent pattern mining algorithms still encounter challenges at mining large and diverse patterns [19]. Therefore, these patterns do not provide significant gains when it comes to Boolean matrix decomposition, and are not powerful in summarizing network processes. An obvious improvement is to rank the patterns by their 'area,' i.e., the products of their frequencies and sizes, and use top-k largest area patterns for decomposition. Even with this modification, frequent subgraph mining methods cannot be applied directly to our netBMF problem, since we would like the frequent subgraphs to be 'diverse' so as to minimize the summarization error.

Boolean Matrix Factorization. Previous work in this class aims at finding a product of matrices that summarizes the binary input data with a smallest possible amount of reconstruction error. The most popular baseline for BMF, ASSO [6], has quadratic time complexity in m, making it not scalable. gDBMF [20] solves a naïve version of BMF, where $M \odot B$ must not contain any extra nonzeros compared to S. To deal with noise and unknown values, Minimum Description Length principles has been used, such as MDL4BMF [21], Nassau [22], PANDA [23], and PANDA+ [24]. Ternary matrix factorization (TMF [25]) considers unknown values and runs in linear time w.r.t. m. TMF has been shown to outperform both ASSO and PANDA+, and is thus chosen as a baseline in our paper. FastStep [26] is a scalable BMF, which relaxes M and B to be real-valued matrices. More recently, [27] performs BMF via message passing to achieve smaller error than ASSO.

BMF methods are the most related ones to our proposed problem, as they share the same objective to summarize binary data. However, their drawback is that they do not consider the network structure present in the data, hence violates condition (2) of netBMF problem. Though, in order to solve the netBMF problem, one can use BMF methods to decompose the input data effectively into two binary matrices and further extract the connected components of each pattern. Modified versions of three state-of-the-art BMF techniques are employed as baseline methods in our experiments.

Finally, graph motifs [28] are repeated network patterns but at different places in the same graph instead of in different graphs as in our problem. NetBMF is also related to finding frequent patterns based on their shapes or the most influential users in information cascades [29] and modelling information cascades [30]. However, we find patterns on real data to reduce the summarization errors instead of building a theoretical cascade model.

Despite all the similarities, there are four features that are considered altogether only by our framework: (a) minimizing the reconstruction error using the pattern set discovered, (b) finding patterns that correspond to the connected subgraphs of the underlying graph, (c) allowing overlapping patterns, and (d) allowing approximate patterns and thus faulttolerance.

2.4 Proposed Solutions

We solve Problem 1 by solving two sub-problems (see Fig. 2.2): Problem 2 finds the mapping M when the state matrix S and the pattern matrix B are fixed, while Problem 3 finds the best patterns B given S using some solution to Problem 2.

Problem 2 (Optimal net-basis mapping) *Given a state matrix $S \in \{0,1\}^{m\times n}$ and a net-basis $B \in \{0,1\}^{k\times n}$, find the optimal mapping matrix $\pi^o_{B,S} \in \{0,1\}^{m\times k}$ to minimize summarization error:*

$$\pi^o_{B,S} = \underset{M\in\{0,1\}^{m\times k}}{\operatorname{argmin}} g(S,B,M) \tag{2.1}$$

Problem 3 (Optimal net-basis) *Given a state matrix $S \in \{0,1\}^{m\times n}$, a solution to Problem 2, find a valid net-basis B to minimize the summarization error as in Problem 1:*

$$B = \underset{B'\in\{0,1\}^{k\times n}}{\operatorname{argmin}} g(S,B',\pi^o_{B',S}) \tag{2.2}$$

Intuitively, given a net-basis B, Problem 2 decides which network states contain which network patterns to minimize the summarization error. In Problem 3, we simply check all possible pattern matrices B' to find the best one B that has the smallest summarization error for a given S (providing that we can find an optimal solution to Problem 2 for each B'). Clearly, such a matrix B is also an optimal solution to Problem 1.

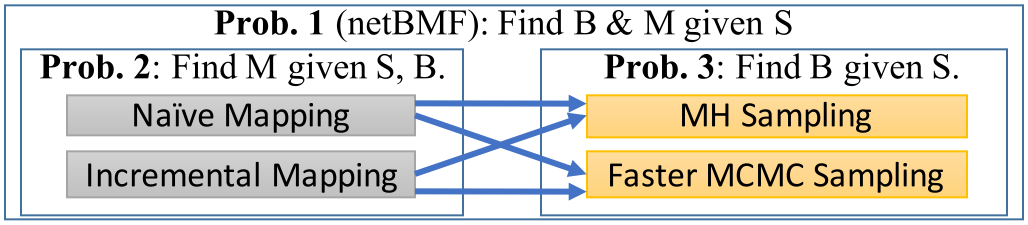

Prob. 1 (netBMF): Find B & M given S
Prob. 2: Find M given S, B.
Naïve Mapping
Incremental Mapping
Prob. 3: Find B given S.
MH Sampling
Faster MCMC Sampling

Figure 2.2: Overview of four solutions for netBMF.

2.4.1 Properties of netBMF problem

We now prove that netBMF problem is *NP-hard* and *not submodular*, which means that we do not currently have a good theoretical bound for a greedy approximate solution to this NP-hard problem as shown in [31].

Theorem 1 *netBMF is NP-hard.*

Proof: BMF (Problem 1 without condition 2) is proved to be NP-hard [6]. Given any instance of the BMF problem, we can always construct an equivalent netBMF problem in polynomial time, where G is a full graph with nodes corresponding to columns of S. Thus, netBMF is also NP-hard. ∎

Theorem 2 *netBMF is not submodular.*

Proof: For brevity, denote $g(B)$ as $g(S, B, \pi^o_{B,S})$. Function g is submodular only if $g(A \cup \{b\}) - g(A) \geq g(B \cup \{b\}) - g(B)$ $\forall$ basis sets $A \subseteq B$ and $\forall$ pattern b. Here, we prove that g is not submodular by a counter example. Consider a full graph with $V = \{1,2,3,4,5,6\}$, $S = [1,1,1,1,0,0]$, and four patterns $G_1 \ldots G_4$, whose node sets are $V_1 = \{1,2,5\}$, $V_2 = \{2,3\}$, $V_3 = \{1,6\}$, $V_4 = \{2,4,6\}$. Choose $A = \{G_1\} \subseteq B = \{G_1, G_2, G_3\}$. It is easy to show that $\pi^o_{A,S} = [1,0,0]$, $\pi^o_{A\cup\{G_4\},S} = [1,0,0,0]$, $\pi^o_{B,S} = [0,1,1]$ or $[1,1,0]$, and $\pi^o_{B\cup\{G_4\},S} = [0,1,1,1]$. Thus, $g(A \cup \{G_4\}) = g(A) = 3$, $g(B) = 4$, and $g(B \cup \{G_4\}) = 5$. Therefore, $g(A \cup \{G_4\}) - g(A) < g(B \cup \{G_4\}) - g(B)$ even though $A \subseteq B$. Thus, g is not submodular and therefore netBMF is not submodular. ∎

2.4.2 Computational challenges

Since Problem 1 (*netBMF*) is NP-hard but not submodular, we cannot bound the quality of a greedy algorithm as proved in [32]. Instead, given a network $G = (V, E)$, a state matrix S and a positive integer k, we need to perform the following costly steps:

(1) enumerate all possible sets of connected nodes (patterns) of V, (2) enumerate all possible sets of k patterns, and (3) solve Problem 2 for each set of k patterns. Intuitively, enumerating all possible connected subsets of V (step 1) is equivalent to enumerating all possible connected subgraphs of G. Because the number of subgraphs of a network grows exponentially with its size, this subgraph space cannot be computed or stored in its entirety. Consequently, both step 1 and step 2 are computationally intractable for large graphs. Moreover, given a pattern matrix $B \in \{0,1\}^{k \times n}$, in order to find the optimal mapping in step 3, we need to check all 2^k possible subsets of B for each network state, leading to a total cost of $O(2^k nm)$ time, where n is the number of nodes, and m is the number of network states. Clearly, this exponential cost is also not scalable for large k.

In our solution, we address the three computational challenges as mentioned above: subgraph enumeration (**challenge 1**), size-k-subset enumeration (**challenge 2**), and fast pattern mapping (**challenge 3**). First, Section 2.4.3 targets ***challenge 2*** with a greedy algorithm. Next, Section 2.4.4 copes with ***challenge 3*** using two linear variants of Problem 2. Finally, Section 2.4.5 deals with ***challenge 1*** by sampling the exponential subgraph search space.

2.4.3 Greedy Algorithm

To tackle ***challenge 2***, we avoid checking all possible sets of k patterns by sequentially adding the pattern with the highest marginal gain to B, as shown in Algorithm 1. The marginal gain $\Delta(B, b)$ is defined as follows.

Definition 2 (Marginal Gain) *Given a state matrix S and a pattern matrix B, the marginal gain $\Delta(B, b) \geq 0$ of a new pattern b is the decrease in summarization error when* *b is added to B, i.e.:*

$$\Delta(B, b) = g(S, B, \pi^o_{B,S}) - g(S, B \cup \{b\}, \pi^o_{B\cup\{b\},S}) \tag{2.3}$$

Algorithm 1 is still not scalable due to two costly tasks corresponding to ***challenges 1*** and ***3*** in Line 3.

Algorithm 1 Greedy_Algorithm(G, S, k)

```
0: $B := \emptyset$
0: **while** $|B| < k$ **do**
0:     $best := \text{argmax}_b\{\Delta(B, b)|$, $b$ is a connected pattern of $G\}$
0:     $B := B \cup \{best\}$
0: **return** $B$ =0
```

2.4.4 Linear net-basis mapping

We cope with ***challenge 3*** by incrementally mapping patterns in B to each network state in S in the order of additions of patterns in Algorithm 1, without updating earlier mappings. Specifically, we propose two variants of Problem 2 that can be solved in linear time w.r.t. k: naïve mapping only allows exact matching, while incremental mapping allows missing nodes.

1) Naïve net-basis mapping: The first variant maps a pattern b to a network state s only if s contains all nodes of b, or more formally:

Problem 4 (Naïve net-basis mapping) *Given a net-basis B with patterns $\{b_1, \ldots, b_k\}$ and a state matrix S with states $\{s_1, \ldots, s_m\}$, a naïve mapping matrix is defined as $\pi^{naive}_{B,S} = M \in \{0,1\}^{m\times k}$, such that $M_{i,q} = 1$ iff $b_q \subseteq s_i$ and $M_{i,q} = 0$ otherwise.*

Theorem 3 *netBMF problem with naïve net-basis mapping, namely netBMFn, is NP-hard but monotonic and submodular.*

Proof: The NP-hard proof is similar to that for netBMF. Monotonicity and submodularity come from the fact that given two net-bases $A \subseteq B$, if a pattern $b \in A$ is mapped to some network state s, then $b \in B$ and will also be mapped to s. ■

2) Incremental net-basis mapping: The naïve mapping does not allow both positive and negative errors in mapping, which renders it less robust to noise. We next introduce the second variant that incrementally maps patterns in B to a network state s as long as this mapping decrease summarization error.

Problem 5 (Incremental net-basis mapping) *Given a net-basis B with patterns $\{b_1, \ldots, b_k\}$ and a state matrix S with states $\{s_1, \ldots, s_m\}$, an incremental mapping matrix is defined as $\pi^{incre}_{B,S} = M^{(k)} \in \{0,1\}^{m \times k}$, where $M^{(0)}$ is empty and $M^{(q)} \in \{0,1\}^{m \times q}$ is the mapping matrix for the first q patterns in B $\forall q \leq k$. Here, $M^{(q)}$ is computed incrementally by adding one new column to $M^{(q-1)}$ such that:*

$$M^{(q)} = \underset{M \in \{0,1\}^{m \times q} s.t. M_{i,j} = M^{(q-1)}_{i,j}, \forall i, \forall j < q}{\text{argmin}} g(S, B, M) \tag{2.4}$$

Theorem 4 *netBMF with incremental net-basis mapping, namely netBMFi problem, is NP-hard and not submodular.*

Proof: Proof is similar to that for netBMF. Please refer to Section 2.4.1. ■

Quality Guarantee: Due to Theorems 2 and 4, we do not have any guarantee on the quality of Algorithm 1 for the optimal netBMF and netBMFi (because g is monotonic but not submodular). However, from Theorem 3, Algorithm 1 guarantees an approximation of $\left(1 - \frac{1}{e}\right)$ or better for netBMFn:

$$g(S, B_{greedy}, \pi^{naive}_{S,B_{greedy}}) \geq (1 - \frac{1}{e}) g(S, B_{opt}, \pi^{naive}_{S,B_{opt}}) \tag{2.5}$$

where B_{opt} is the optimal solution of netBMFn, and B_{greedy} is the greedy solution by

Algorithm 1 that iteratively adds the pattern with the maximum marginal gain [32]. Moreover, no other polynomial time algorithm can achieve an approximation guarantee better than $1 - \frac{1}{e}$ unless $P = NP$, as proved in [31].

Running time: Running time of both net-basis mapping variants is $O(mnk)$, where m, n, k are the number of states, nodes, and patterns respectively. This is because for each network state, we need to compute its matching against each of the k patterns, which entails iterating through all nodes in the worst case.

2.4.5 Incremental sampling of net-basis

To deal with ***challenge 1***, by leveraging the given network structure, we explore the subgraph space of G to sample the subgraphs with high marginal gains. Doing so has two main advantages; (1) it makes sure the *connectivity* constraint of netBMF holds by eliminating disconnected patterns from the search space, and at the same time, (2) it makes our solution *scalable* which helps combatting NP-hardness. Later, a sampled *set* of subgraphs, denoted by $\mathcal{B}$, are used to construct a pattern *matrix* B. Figure 2.3 illustrates the relation between G, B and $\mathcal{B}$. For the remaining of this section, to distinguish one from another, we use the term **basis subgraph** for each element in $\mathcal{B}$, and **pattern** for a set of non-zero elements in each row of B.

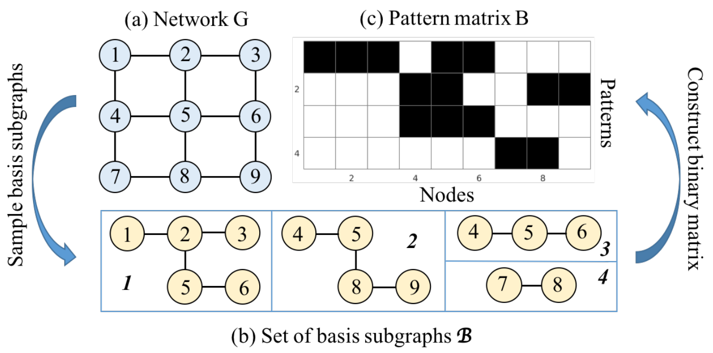


Figure 2.3: (a) Grid network G, (b) Basis subgraphs set $\mathcal{B}$ and (c) Net-basis (Pattern) Matrix B

We first introduce the high-level sampling-based netBMF algorithm in Section 2.4.5.1. After that, we propose how to sample the subgraph space in Section 2.4.5.2 and 2.4.5.3. Finally, Section 2.4.5.4 describes how to choose a good starting point for sampling.

Algorithm 2 netBMF($G, S, k, numSeeds, maxIter$)

```
1: $\mathcal{B} \leftarrow \emptyset$
2: while $|\mathcal{B}| < k$ do
3:     $best := \emptyset$
4:     $seeds :=$Get_Seed_Nodes($G, S, \mathcal{B}, numSeeds$)
5:     for $seed \in seeds$ do
6:         $b :=$ Sample_Basis($G, S, \mathcal{B}, seed, maxIter$)
7:         $b^* :=$ Improve_Basis($G, S, \mathcal{B}, V_q$)
8:         if $\Delta(\mathcal{B}, b^*) > \Delta(\mathcal{B}, best)$ then
9:             $best := b^*$
10:        end if
11:    end for
12:    $\mathcal{B} \leftarrow \mathcal{B} \cup \{best\}$
13: end while
14: $B \leftarrow \mathcal{B}$
15: return $B$ =0
```

2.4.5.1 Sampling-based netBMF algorithm

Algorithm 2 is the high-level sampling-based netBMF algorithm which incorporates sampling into Algorithm 1. We start with an empty set $\mathcal{B}$ in Line 1, and then greedily add the subgraphs with the highest marginal gains to $\mathcal{B}$ until we obtain k basis subgraphs in Lines 2-10. In each iteration, we first get a list of seed nodes for sampling in Line 4 (detailed in Section 2.4.5.4), then perform sampling from these seed nodes in the subgraph search space to get a good candidate subgraph in Line 6, and further improve this candidate in Line 7 (Algorithms 3 and 4 in Section 2.4.5.2). The best sampled subgraph is recorded in Lines 8-9, and later added to $\mathcal{B}$ in Line 10.

2.4.5.2 MH-sampling in subgraph space

In Line 6 of Algorithm 2, we need the undirected connected subgraphs of G with the highest marginal gain (Equation 2.3). We thus utilize the Metropolis-Hastings algorithm to sample subgraphs with probability proportional to their marginal gains.

Edit graph (EG): To systematically navigate the subgraph space, we formulate it as an edit graph, which represents all possible edits that can be performed on any basis subgraph b. We denote $b \to u$ and $b \leftarrow u$ as the new connected subgraphs obtained by removing and adding node u to a connected subgraph b respectively.

Definition 3 (Edit Graph) *The edit graph (EG) of a network $G = (V, E)$ is a directed edge-weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$, where $\mathcal{V} = \{b | b$ is an undirected connected subgraph of $G\} \cup \{\emptyset\}$, $\mathcal{E} = \{(b, b') | b' = b \leftarrow u$ or $b' = b \to u, u \in V, b \in \mathcal{V}, b' \in \mathcal{V}\}$, and $\mathcal{F} : \mathcal{E} \to \mathbb{R}$ is a function that assigns weights to edges in $\mathcal{E}$.*

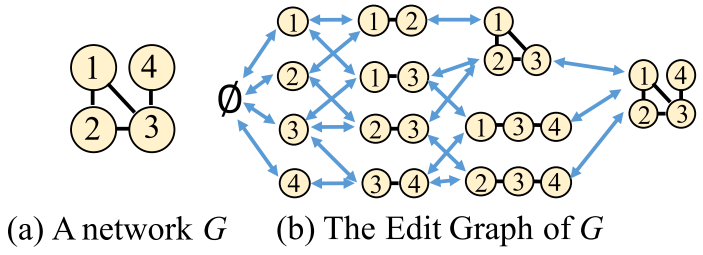


Figure 2.4: (a) An example network G and (b) its Edit Graph $\mathcal{G}$.

Fig. 2.4 shows an example EG. The EG is simply a graph whose vertices are undirected connected subgraphs of G or the empty subgraph. For clarity, we use the term "node" for the network G, and "vertex" for the EG. An edge between two vertices in the EG indicates that we can edit the corresponding subgraphs into each other by adding or deleting one node. The edge weights reflect the potential impact of the edits on the marginal gain and are explained in Section 2.4.5.2. Since G is finite, the EG is also a *finite* space. Its size is *exponential* w.r.t. the size of G, making it infeasible to be computed or stored in its entirety. More importantly, the EG is a *connected graph*, thanks to the inclusion of the empty subgraph $\emptyset$. As a result, we can start at a random subgraph and, after a finite series of *local edits*, reach any other desired subgraph. We next exploit this property to perform sampling in the EG.

Metropolis-Hastings (MH) sampling in EG: Using the MH algorithm, we can sample a subgraph b with probability approximately proportional to its marginal gain. In other words, the *target visit probability distribution* is τ, where

$$\tau_i = \frac{w_i}{\sum_{a_i \in \Omega} w_i} \tag{2.6}$$

$$w_b = \Delta(B, b) + 1 \tag{2.7}$$

Here, the addition of 1 is to avoid division by zero in Eqn. 2.6.

MH algorithm simulates τ by a Markov chain in the EG with an arbitrary *proposal distribution matrix* Q. If the current state X_t at time step t is subgraph b_i, MH algorithm finds the next state X_{t+1} as follows:

- Draw a random subgraph b_j with probability Q_{ij}.
- Compute the *acceptance probability* α_{ij}:

$$\alpha_{ij} = min\left\{1, \frac{\tau_j Q_{ji}}{\tau_i Q_{ij}}\right\} = min\left\{1, \frac{w_j Q_{ji}}{w_i Q_{ij}}\right\} \tag{2.8}$$

- $X_{t+1} = \begin{cases} b_j & \text{with probability } \alpha_{ij} \\ b_i & \text{with probability } 1 - \alpha_{ij} \end{cases}$

The Markov chain as described above is *reversible*, *ergodic*, and satisfies the detailed balance property. Thus, the stationary distribution of this Markov chain is unique and to converge to the target distribution τ in Eqn. 2.6.

Proposal distribution matrix: In our solution, Q contains the edge weights in the EG, which reflect the "quality" of the edits on basis subgraphs. A "good" edit increases our chance of finding basis subgraphs with high marginal gains. Since it is costly to compute the exact marginal gains for all neighbors of a basis subgraph b in the EG, we approximate the quality of edits by a *potential vector* $\mathbf{y}$. Given the current set of basis subgraphs $\mathcal{B}$, net-basis matrix B, mapping matrix M, and the current candidate basis subgraph b to be added to $\mathcal{B}$, we define $\hat{S}$ as the part of the state matrix S that has not been covered by any basis subgraphs in $\mathcal{B}$, i.e.,

$$\hat{S}_{i,j} = \begin{cases} 0 & \text{if} \quad (M \odot B)_{i,j} = 1 \\ S_{i,j} & \text{otherwise} \end{cases} \tag{2.9}$$

In addition, denote $\mathbf{x} \in \{0,1\}^{m\times 1}$ as the mapping of b to S, i.e., $\mathbf{x}$ is the last column of $\pi_{\mathcal{B}\cup\{b\},S}$: $x_i = 1$ iff state s_i is mapped to b. Finally, we define the *potential vector* $\mathbf{y} \in \{0,1\}^{1\times n}$ as follows:

$$\mathbf{y} = \mathbf{x}^T \times \hat{S} \tag{2.10}$$

Intuitively, y_j counts how many times a node v_j in G is contained in the network states that b is mapped to, excluding the portion of S that has already been covered by the current set $\mathcal{B}$. If y_j is high, then node v_j belongs to the same network states with subgraph b many times. In this case, adding v_j to b potentially increases the marginal gain, while removing v_j from b (if $v_j \in b$) potentially decreases the marginal gain. Denote A and D as the sets of connected basis subgraphs that can be obtained by adding and removing one node from b respectively:

$$A = \{b' = b \leftarrow v_j | v_j \in V\} \tag{2.11}$$

$$D = \{b' = b \rightarrow v_j | v_j \in V\} \tag{2.12}$$

We now define the proposal distribution matrix Q as follows:

$$Q_{bb'} = \begin{cases} \beta \frac{1}{C_A}(y_j + \epsilon) & \text{if} \quad b' = b \leftarrow v_j \in A \\ (1-\beta)\frac{1}{C_D}\frac{1}{y_j+\epsilon} & \text{if} \quad b' = b \rightarrow v_j \in D \end{cases} \tag{2.13}$$

where $\beta \in [0,1]$ is the probability of addition, $0 < \epsilon \ll 1$ is a small constant to avoid division by zero, C_A and C_D are two normalization constants defined as $C_A = \sum_{b'=b\leftarrow v_j \in A}(y_j + \epsilon)$ and $C_D = \sum_{b'=b\rightarrow v_j \in D} \frac{1}{y_j+\epsilon}$. Based on Eqn. 2.13, we perform an addition with probability proportional to the potential of the added node, while a deletion with probability inversely proportional to the potential of the removed node. The role of β is to account for the fact that the number of supergraphs of a basis subgraph b (which is equal to the number of neighbors in G of the nodes in b) is likely to be significantly larger than the number of its subgraphs (which is at most its size $|b|$). Thus, without β, it is not likely for a deletion to be chosen in the sampling process. Note that the sampler would also explore some "bad" edits to avoid being stuck in local optima.

Final MH algorithm in the EG: The MH algorithm for sampling the next basis subgraph with high marginal gain is shown in Algorithm 3. Here, the sampler starts at a given seed node, performs sampling in this node's neighborhood for at most *maxIter* steps, and returns the visited basis subgraph with the highest marginal gain. Due to the randomness of sampling and the limitation of *maxIter*, it is likely that the basis subgraph returned by Algorithm 3 can still be further improved locally. Hence, we propose to greedily perform the best possible local edits on this basis set until no more good edits can be found, as shown in Algorithm 4.

Algorithm 3 Sample_Basis($G, S, \mathcal{B}, seed, maxIter$)

```
1: $t := 0$
2: $X_t := \{seed\}$
3: Compute $\pi_{\mathcal{B}\cup\{X_t\}}(\mathcal{S})$ and $\Delta(\mathcal{B}, X_t)$
4: $best := X_t$
5: while $t < maxIter$ do
6:     $A := \{X_t \leftarrow u | u \notin X_t, \exists u' \in X_t, (u, u') \in E\}$
7:     $D := \{X_t \rightarrow u | u \in X_t, u$ is not a cut-vertex$\}$
8:     Compute $Q_{X_t b}$, $\forall\ b \in A \cup D$ based on Eqn. 2.13
9:     Choose a neighbor $b$ from proposal distribution $Q_{X_t b}$
10:    Compute $w_{X_t}$ and $w_b$ based on Eqn. 2.7 and $Q_{bX_t}$ based on Eqn. 2.13
11:    $\alpha := \frac{w_b Q_{bX_t}}{w_{X_t} Q_{X_t b}}$
12:    if $uniform(0, 1) \leq \alpha$ then
13:        $t := t + 1$
14:        $X_t := b$
15:        if $\Delta(\mathcal{B}, X_t) > \Delta(\mathcal{B}, best)$ then
16:            $best := X_t$
17:        end if
18:    end if
19: end while
20: return $best$ =0
```

Algorithm 4 Improve_Basis($G, S, \mathcal{B}, b$)

```
1: Compute $A^+$ and $D^+$ for $b$
2: while $A^+ \cup D^+ \neq \emptyset$ do
3:     $b := argmax_{b' \in A^+ \cup D^+} \Delta(\mathcal{B}, b')$
4:     Update $A^+$ and $D^+$ for $b$
5: end while
6: return $b$ =0
```

2.4.5.3 Faster MCMC algorithm

Since we only care about a single basis subgraph with the highest marginal gain at a time, we do not necessarily need to sample subgraphs based on the exact target distribution in Eqn. 2.6. Therefore, to reduce running time, we adopt a new acceptance probability instead of Eqn. 2.8:

$$\alpha_{ij} = min\,\{1, w_j/w_i\} \tag{2.14}$$

Here, we always accept a good move that increases marginal gain. Otherwise, the acceptance probability is equal to the ratio of the marginal gains of two basis subgraphs.

As a result, we avoid computing Q_{bX_t} in Line 10, saving considerable computational cost, and replace Line 11 with Eqn. 2.14 in Algorithm 3. More importantly, the modified MCMC algorithm is still *finite, reversible, ergodic, and thus converges to a unique stationary distribution* even though we do not have a closed form of this distribution as in Eqn. 2.6 for MH.

2.4.5.4 Choosing seeds for sampling

In practice, we cannot run the MH or MCMC algorithm for an infinite number of iterations to converge to the stationary distribution. As a consequence, it is more practical to start the sampler where we are more likely to find basis sets with high marginal gains. We sample seed nodes with probability proportional to their frequencies in $\hat{S}$ in Section 2.4.5.2, i.e., the number of nonzeros in the corresponding columns of $\hat{S}$.

2.4.5.5 Running time

A conservative upper bound for Algorithm 3 (for both MH and MCMC) is $O(mnkI + \hat{d}I^2)$, where I is $maxIter$, and $\hat{d}$ is the maximum node degree in G. In particular, in each iteration of the while loop from line 5 to line 16, the number of nodes in pattern X_t can be increased by up to 1 node. Thus at the end of this while loop, X_t can contains up to I nodes. The most costly operations in this while loop are line 7 and line 10. In line 7, we need to find all cut vertices of a subgraph X_t, which can be done in $O(n_{X_t} + e_{X_t})$, where n_{X_t}, e_{X_t} are the number of nodes and edges in X_t respectively, and are upper-bounded by I and $\hat{d}I$ respectively. The computation in line 10 is overpowered by Eqn. 2.7, which costs $O(mnk)$. Thus, after $maxIter = I$ iterations, the upperbound for complexity is $O(I * (mnk + \hat{d}I))$.

2.5 Experiments

We compare four variants of netBMF algorithm (Figure 2.2) to other baselines in four datasets. In summary, NetBMF provides better summarization than BMF (ASSO, TMF, and PANDA+) and frequent subgraph mining (gSpan), with more interpretable patterns.

2.5.1 Settings

Synthetic data: We generate a network of 1000 nodes (average degree = 10) using the Barabasi-Albert algorithm [33]. We sample 20 random ground-truth network patterns, and then delete or add 5 random nodes in each of them to create 100 network states in total.

Real data: We use three real world datasets: (1) **Youtube** [34], where each network state is a community of users who joined the same group, and the graph represents users' friendship; (2) **DBLP**, where we generate network states as lists of all attendees at scientific conferences, and the graph represents their co-authorship (using raw data from [35]); (3) **Traffic** [36], where each node is a road segment, each network state contains the list of congested road segments (when the speed is in the bottom 5 percentile of speeds on this segment), and the graph captures the adjacency of road segments. Table 2.2 summarizes the datasets.

Baselines: We compare netBMF with two groups of baselines: (i) top-k frequent subgraph mining with best frequency (gSpan-Freq) or coverage (gSpan-Cover) among the

Dataset	Youtube	DBLP	Traffic	Synthetic
#nodes (n)	1157828	7916	100	1000
#states (m)	5000	219	8640	100
#edges ($\|E\|$)	5975248	75067	256	39200
$\|\|S\|\|_F^2$	72959	72138	43768	2078

Table 2.2: Statistics of real-world datasets.

top-1000 frequent subgraphs returned by gSpan [15]; (ii) Boolean Matrix Factorization, including ASSO [6], TMF [25], and PANDA+ [24]. We study four variants of netBMF based on pattern mapping approach (incremental or naïve) and sampling approach (MH or MCMC): netBMFi-MH, netBMFi-MC, netBMFn-MH, and netBMFn-MC.

Default parameter settings: For the upcoming experiments, we use $\beta = 0.7$, $numSeeds = 10$, $maxIter = 200$ as default parameters. The maximum number of patterns k for Youtube and DBLP is 100, and for Traffic is 30. Note that TMF and PANDA+ often fail to find enough k patterns as requested. We experiment the running times of our methods on the synthetic dataset with parameters $n = 1000$, $m = 100$, and $k = 10$. During these experiments, we change one parameter at a time, while keeping the others fixed. The result of each experiment is averaged over 5 runs of netBMF. Experiments were run on Ubuntu with Intel Core i7-5930K CPU, 3.50GHz. Code is written in Python.

We use the relative error to evaluate the decomposition:

$$RelativeErr(S, M, B) = \frac{||S \oplus (M \odot B)||_F^2}{||S||_F^2} \times 100\% \tag{2.15}$$

To check the *stability of sampling*, we use Jaccard index. Given two net-bases B_1 and B_2, denote n_{11} as the number of node pairs that are in the same basis subgraphs in both B_1 and B_2, n_{10} as the number of node pairs that are in the same basis subgraphs in B_1 but not in B_2, n_{01} as the number of node pairs that are in the same basis subgraphs in B_2 but not in B_1. Then, the Jaccard similarity coefficient (or Jaccard index) between B_1 and B_2 is defined as:

$$J(B_1, B_2) = \frac{n_{11}}{n_{01} + n_{10} + n_{11}} \tag{2.16}$$

Clearly, $J(B_1, B_2) \in [0, 1]$. $J(B_1, B_2) = 1$ means B_1 and B_2 are identical, while $J(B_1, B_2) = 0$ means they do not overlap.

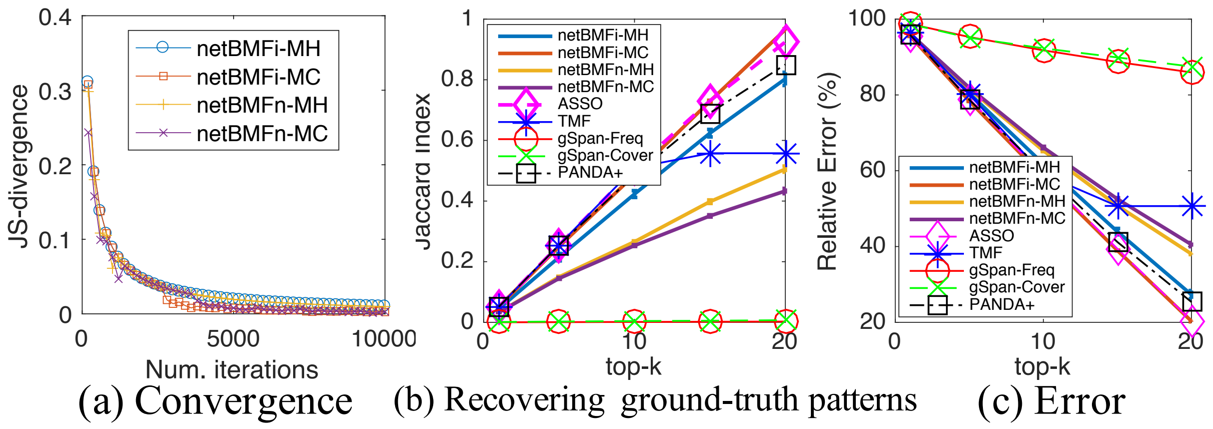


Figure 2.5: Synthetic data: (a) Convergence of sampling; (b) Jaccard index between obtained patterns and the 20 ground-truth patterns; (c) Summarization error.

2.5.2 Convergence and stability of sampling

Our algorithms produce reasonably stable results on the synthetic dataset despite the huge exponential search space with many different subgraphs having the same marginal gains.

Convergence of sampling: All of our sampling algorithms converge to a unique stationary distribution (Section 2.4.5.2), which is confirmed experimentally in Fig. 2.5a. Here, the Jensen-Shannon (JS) divergence compares the visit distribution at every 200 iterations with the previously computed one. All variants of netBMF converge quickly after 2000 iterations.

Recovering ground-truth synthetic patterns: Our algorithms consistently recover the 20 ground-truth patterns in the synthetic dataset at $k = 20$, with comparable or smaller errors than other baselines as reported. Fig. 2.5b figure shows the Jaccard index between the obtained net-basis patterns and the ground-truth patterns in the synthetic dataset after 10 runs of netBMF, compared with other baselines. We observe similar results for the relative errors in Fig. 2.5c. Overall, netBMF*i*-MC is the best at recovering the ground truth.

Stability of sampled net-bases: It must be noted that the error bars for 5 different runs of all versions of netBMF are very small (almost invisible) in Fig. 2.5b and 2.5c, suggesting that netBMF has high stability.

2.5.3 Summarization error

Table. 2.3 shows the relative errors for different baselines when $k = 100$, where for all three datasets, the top-2 smallest errors are from netBMF variants. Both versions of gSpan perform poorly since they retrieve very small subgraphs of high frequencies. For fair comparison, we also extracted only the connected subgraphs with the highest coverage from the patterns found by the BMF baselines (the modified connected version of BMFs). In general, when connected patterns are taken into account, netBMF variants provide much smaller errors compared to BMF (ASSO, TMF, PANDA+) and frequent subgraph mining (gSpan) methods. Even if the connectedness constraint is ignored, the errors of netBMF are still comparable and often smaller than the original version of BMFs. This reveals that netBMF methods can also be applied to classic BMF problems, if the data is large and known to have an underlying network structure.

Table. 2.4 further shows the relative errors as k varies. Overall, all methods have a decreasing trends in errors. gSpan clearly has very high errors regardless of k. TMF performs well when k is small since it simply picks the whole network states as patterns when the network states are big, while these network states often have low frequencies and are disconnected. Thus, as k increases enough to cover the big datasets (Youtube and DBLP), netBMF variants gradually outperform TMF. PANDA+ returns high errors, especially in the case of Traffic, where the relative errors are even greater than 100%, suggesting that PANDA+ focuses more on frequent patterns rather than accurate patterns. ASSO has low and decreasing errors as k increases. However, this comes at a huge cost in running time as we explain in Section 2.5.5. Overall, netBMF variants have lower errors than most baselines as k varies.

Variants of netBMF: Given the relative errors in Table. 2.4, there is no clear winner among four netBMF variants. On average, MH sampling seems to provide smaller errors compared to MCMC sampling, at the expense of longer running time as shown later. It is also surprising that the incremental mapping does not consistently outperform naive mapping. Thus, we next consider other factors (e.g., pattern quality and running time) to decide which method to use in practice.

2.5.4 Quality of net-bases

We evaluate the quality of the basis subgraphs (or induced subgraphs for ASSO, TMF, and PANDA+) with a number of metrics: size (i.e., the average number of nodes); frequency; connectedness (i.e., clustering coefficients); and the number of found patterns $\overline{k}$. Overall, with the goal of low summarization errors as the highest priority, netBMF obtains connected, large, frequent, and more patterns than its counterparts.

Table. 2.3 reports the results averaged over 100 basis subgraphs for different baselines. NetBMF variants outperform all other baselines w.r.t. the above metrics, i.e., the obtained basis subgraphs are big (high number of nodes), connected, and dense (high clustering coefficients). On the contrary, the original implementation of TMF, ASSO, and PANDA+ find big but disconnected patterns with extremely low clustering coefficients. Note that TMF, ASSO, and PANDA+ may fail to find enough k patterns, thus their averages are only computed on the non-empty patterns. That being said, if these empty patterns (size, frequency, and clustering coefficient are all zeros) are taken into consideration, the average results for BMF methods will be much lower. Additionally, the connected versions of TMF, ASSO, and PANDA+ may find patterns that are big and frequent, but with either

Method	**Youtube** (max $k = 100$)				
	Error (%)	*Size*	*Freq.*	*C.C.*	$\overline{k}$
netBMFn-MH	**79.5**	145.9	2.0	0.075	100
netBMFi-MH	87.0	62.2	7.8	0.196	100
netBMFn-MC	86.6	76.0	8.1	0.199	100
netBMFi-MC	**83.9**	85.6	4.2	0.135	100
gSpan-Cover	99.7	4.8	28.6	0.752	100
gSpan-Freq	98.6	2.8	35.9	0.727	100
TMF (connected)	85.7	522.2	1.8	0.011	19
PANDA+ (connected)	92.2	19.8	8.8	0.283	100
ASSO (connected)	-	-	-	-	-
TMF	85.7	661.4	1.3	0.013	15
PANDA+	86.9	38.9	6.2	0.165	100
ASSO	-	-	-	-	-

Method	**DBLP** (max $k = 100$)				
	Error (%)	*Size*	*Freq.*	*C.C.*	$\overline{k}$
netBMFn-MH	**62.4**	271.9	1.0	0.024	100
netBMFi-MH	87.1	38.3	13.4	0.288	100
netBMFn-MC	**75.1**	183.2	1.0	0.034	100
netBMFi-MC	82.7	77.2	5.8	0.107	100
gSpan-Cover	99.8	75	2.0	0.050	100
gSpan-Freq	95.9	1.4	39.5	0.280	100
TMF (connected)	84.8	97.9	5.7	0.188	100
PANDA+ (connected)	94.7	9.0	21.7	0.414	100
ASSO (connected)	76.0	273.7	2.8	0.024	100
TMF	83.1	748	1.5	0.008	15
PANDA+	93.4	51.2	17.2	0.103	32
ASSO	72.0	345.6	2.3	0.013	100

Method	**Traffic** (max $k = 30$)				
	Error (%)	*Size*	*Freq.*	*C.C.*	$\overline{k}$
netBMFn-MH	51.7	2.8	316.8	0.536	30
netBMFi-MH	**47.6**	5	397.4	0.238	30
netBMFn-MC	51.8	2.9	314.4	0.528	30
netBMFi-MC	**47.6**	5	397.4	0.238	30
gSpan-Cover	75.8	5.5	232.2	0.395	30
gSpan-Freq	74.1	1.1	421.4	0.100	30
TMF (connected)	61.3	6.3	247.1	0.321	30
PANDA+ (connected)	143.8	2.9	342.2	0.320	30
ASSO (connected)	62.5	1.3	438.8	0.222	30
TMF	56.4	18.3	159.4	0.102	15
PANDA+	151.5	18.2	241.7	0.067	6
ASSO	56.5	1.5	436.8	0.190	30

Table 2.3: Comparing baselines with three real-world datasets. Top-2 smallest errors for connected patterns are shown in bold. The connected (*con.) versions of TMF, PANDA+, and ASSO only contain the top connected subgraphs with highest coverage extracted from their corresponding patterns (similar to gSpan-Cover). TMF and PANDA+ may not find enough k patterns. ASSO failed to finish in 4 days for Youtube. (C.C. is clustering coefficient)

	Youtube					**DBLP**					**Traffic**				
k	**10**	**30**	**50**	**80**	**100**	**10**	**30**	**50**	**80**	**100**	**5**	**10**	**15**	**20**	**30**
netBMFn-MH	**96.1**	**90.3**	**86.8**	**81.4**	**79.5**	91.4	**84.1**	**76.7**	**67.6**	**62.4**	86.7	76.4	68.2	62.2	51.7
netBMFi-MH	97.9	94.0	92.3	88.2	87.0	94.9	92.1	90.5	88.3	87.1	**79.6**	**64.8**	**58.0**	**53.0**	**43.3**
netBMFn-MC	98.0	94.1	90.9	87.9	86.6	95.0	89.1	84.5	78.3	**75.1**	86.3	76.0	68.1	62.2	51.8
netBMFi-MC	96.6	91.7	89.0	**85.4**	**83.9**	93.4	90.1	87.5	84.6	82.7	**76.4**	**67.8**	**62.2**	**57.1**	**47.6**
gSpan-Cover	99.8	99.8	99.7	99.7	99.7	99.8	99.8	99.8	99.8	99.8	95.0	91.5	87.1	85.4	75.8
gSpan-Freq	99.5	98.8	98.7	98.7	98.6	99.3	98.1	97.2	96.4	95.9	96.0	92.0	87.2	83.4	74.1
TMF(con.)	**88.4**	**85.7**	**85.7**	85.7	85.7	**89.1**	85.2	85.0	84.9	84.8	85.2	76.0	70.6	67.4	61.3
PANDA+(con.)	96.9	94.5	93.3	92.5	92.2	95.5	95.0	94.8	94.7	94.7	120.0	127.6	132.2	136.6	143.8
ASSO(con.)	-	-	-	-	-	**91.6**	**84.3**	**80.3**	**76.9**	76.0	89.6	82.9	77.7	72.6	62.5

Table 2.4: Relative error (%) for different k. Top-2 smallest errors for each k are shown in bold.

high errors or fewer number of patterns than the provided k. gSpan yields connected but mostly smaller subgraphs. As a result, netBMF outperforms all other baselines in summarizing network processes.

The choice of MH or MCMC sampling does not affect the quality significantly, whereas for large k (Youtube and DBLP), netBMFn tends to obtain bigger basis subgraphs than netBMFi on average, which is at first quite surprising given that netBMFi allows more error in basis mapping. However, the actual reason behind this behaviour is that, initially, netBMFi greedily exploits the search space by returning much larger patterns than netBMFn. This causes the later patterns to be smaller, and as a result, the average size of the patterns returned by netBMFn can be larger than netBMFi. This behaviour of netBMFi can be restricted by lowering the parameter β. For the same reason, the clustering coefficients of netBMFn are also higher than those of netBMFi in Youtube and DBLP, but smaller in Traffic. We conjecture that the performance of netBMF variants will depend on the number of patterns k and the specific datasets, i.e., the connectivity in G, and the network states in S.

Use case: Following the idea illustrated in Fig. 2.1, we now demonstrate the potential of the discovered patterns by our framework. We employ a large dataset combining mobility traces of taxi cabs in San Francisco and road network extracted from OpenStreetMap [37, 38]. The first one consists of GPS trajectories from 500 cabs over 30 days in San Francisco, mapped to the corresponding road segments on the road network. We consider every 30 min period as one *ride*, with a condition of a cab travelling more than a mile during this period. As a result, a ride passes through 112 road segments on average. Later, we regard each cab ride as a unique network state, and construct our binary state matrix S, where $S_{q,i} = 1$ if a ride q passes through road segment i. The resulting S matrix has 362434 rows, and 78847 columns, whereas the road network G has 78847 nodes and 181598 edges.

Fig. 2.6 further shows the top-12 mined patterns from S by our netBMFi-MH variant. Each pattern is a set of connected road segments that highlights some touristic attractions or highways with high volume of traffic. Note that these patterns also cover different parts of the city, suggesting that they efficiently summarize the taxi cab trajectories.

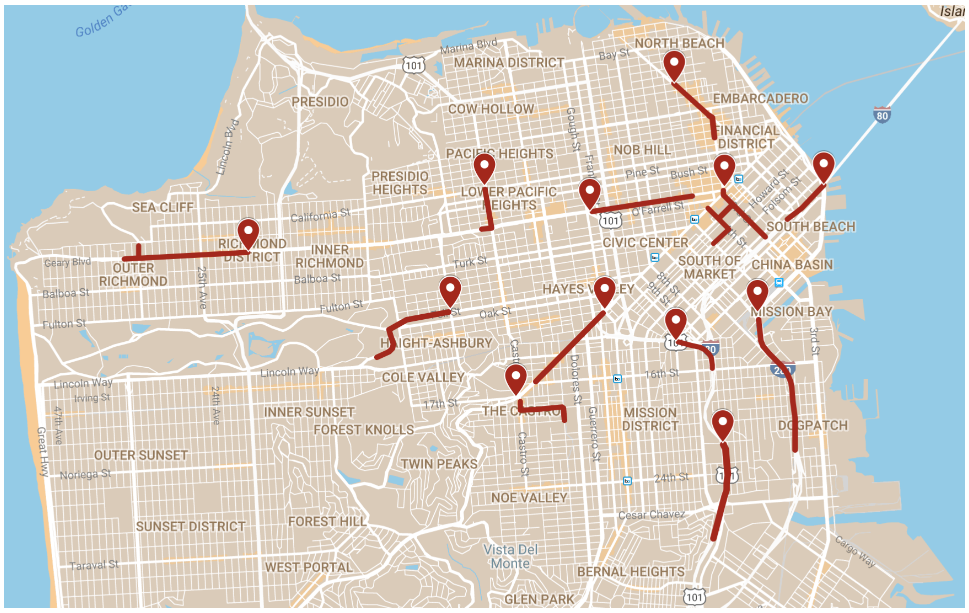


Figure 2.6: netBMFi-MH patterns (red lines) pinpoint the popular attractions and main highways in San Francisco.

2.5.5 Running time

All four variants of netBMF have linear running time in k, n, m, and $numSeeds$, as shown in Fig. 2.7b-e. We further evaluate the speed of three different net-basis mappers

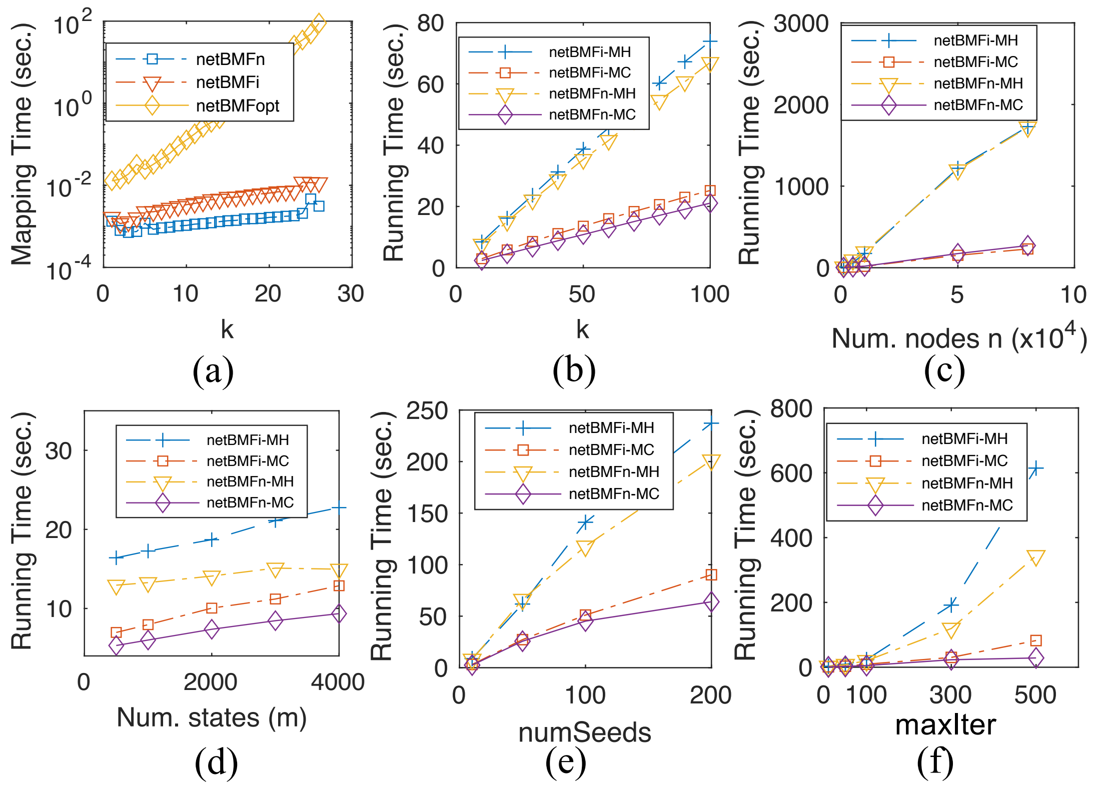


Figure 2.7: Running time for four variants of netBMF as their parameters are varied. For all num. states in (g): $||S||_F^2/m/n = 0.1$.

on a single process state as the number of patterns k increases in Fig. 2.7a. As discussed in Section 2.4.2, the running time increases linearly in k for netBMFi and netBMFn, but exponentially for netBMFopt, making the optimal solution not scalable. Fig. 2.7f confirms that the running time is quadratic in $maxIter$ (Section 2.4.5.5). A higher number of iterations likely leads to bigger visited subgraphs, which potentially have more neighbors in the EG and cause more computational cost for the transition probability Q in Lines 8 and 10 of Algorithm 3. Finally, MH sampling is slower than MCMC sampling due to the computation of the transition probability $\mathcal{Q}_{X_t b}$ in Line 10 of Algorithm 3. MCMC sampling discards this computation and uses $\alpha = w_b/w_{X_t}$ instead in Line 11, reducing computational cost significantly. To put this into perspective, ASSO on Youtube dataset did not finish after 4 days, TMF took approximately 51 hours, while our netBMFn-MC variant took less than 7 hours to complete. PANDA+ is the fastest (less than 1 hour on Youtube), yet it achieves the worst summarization error among all the other BMFs.

2.6 Conclusion

In this chapter, we proposed a novel problem of summarizing network processes by mining coherent subsets of network nodes that often engage in similar network processes. To cope with its NP-hardness, we designed a greedy algorithm and four scalable variants that effectively sample the exponential subgraph space to find the best network patterns. Extensive experiments demonstrate our solution to be efficient in mining relevant and accurate network patterns in both synthetic and real datasets. Promising future directions include modeling the dynamic behavior of network processes using the discovered patterns, and using directed subgraphs as network patterns.

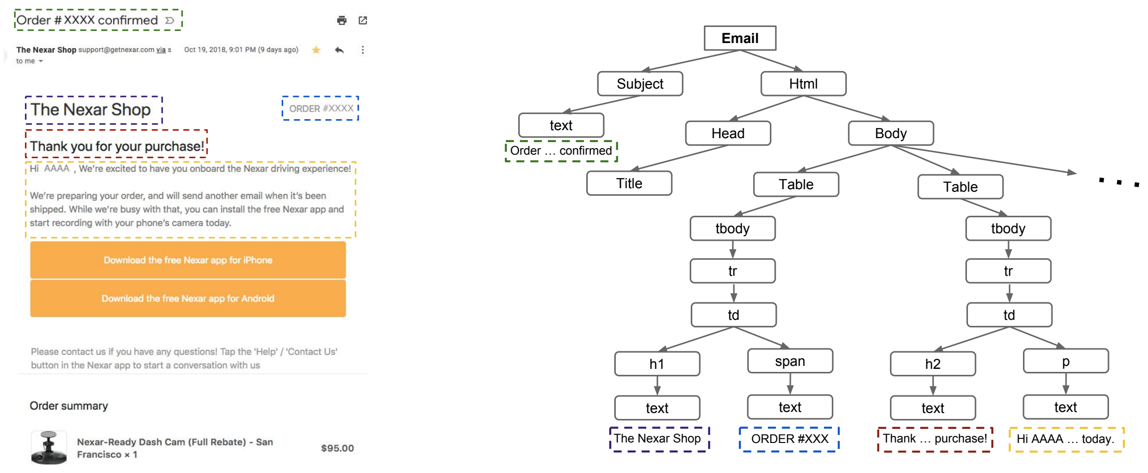
Chapter 3

RiSER: Learning Better Representations for Richly Structured Emails

3.1 Introduction

Classifying documents into two or more target classes is a core problem in many web applications, and has received attention from several research communities. Identifying the sentiment in text documents [39], detecting spam in email [40] and the web [41], and fake news detection [42] are well-known applications. Even enterprise applications like legal discovery [43] make use of the core abstraction of document classification.

Most existing approaches focus on short plain-text documents like tweets, reviews, and posts in online discussion forums. However, several interesting applications are on corpora where the documents are longer, and often have rich HTML markup structure. Such corpora have not received as much research attention, but are important for many applications. For instance, web pages have rich HTML structure; classifying them [44] is


Order # XXXX confirmed

The Nexar Shop support@getnexar.com via s Oct 19, 2018, 9:01 PM (9 days ago)
to me

The Nexar Shop ORDER #XXXX

Thank you for your purchase!

Hi AAAA , We're excited to have you onboard the Nexar driving experience!

We're preparing your order, and will send another email when it's been shipped. While we're busy with that, you can install the free Nexar app and start recording with your phone's camera today.

Download the free Nexar app for iPhone

Download the free Nexar app for Android

Please contact us if you have any questions! Tap the 'Help' / 'Contact Us' button in the Nexar app to start a conversation with us

Order summary

Nexar-Ready Dash Cam (Full Rebate) - San Francisco × 1 $95.00

Figure 3.1: An example *purchase* email (left) and a simplified DOM tree (right) representing a portion of the email. For clarity, we trim long trivial HTML sequences such as nested *div* tags. Note that we extend the DOM tree beyond the email message body to include a branch representing the subject.

useful for several applications like focused crawling, faceted search, and even as a signal for web page search ranking.

Email is another area that contains longer documents with rich HTML markup structure. Recent studies [45] have shown that most email is richly formatted and machine-generated rather than plain-text sent by humans. Several interesting applications beyond spam-detection rely on learning good classifiers over email. This includes foldering [46], automatic prioritization [47], and even information extraction [48, 49, 50]. In particular, these studies show that learning a high-precision classifier over richly formatted emails is a key ingredient to extracting structured data from email that is then used to power several intelligent experiences [49, 48].

In this paper, we tackle the problem of learning a good representation (embedding) for structured documents like richly formatted emails. We focus on evaluating these embeddings for the task of classifying an email into one of k target classes. Recent literature has shown that for relatively short documents, a recurrent neural network with an attention mechanism [51] is a very strong baseline for learning good representations. With richly formatted documents, most existing techniques ignore the formatting information except for assuming that documents are hierarchically organized [51] into sentences and paragraphs. We illustrate in Section 3.2 that the rich structure of HTML-formatted email can be a valuable signal for the relative importance of various pieces of information in the document. We introduce a neural architecture called RiSER (Richly Structured Email Representation) to take advantage of the structure of the email in addition to the content. RiSER projects the input email into a vector representation by jointly encoding the HTML structure of the email along with the terms in the email. It consists of a 2-level LSTM to first construct a structural encoding and combines this with an encoding of the terms for which there is a second LSTM layer. This provides a simple way to encode the formatting and layout information as opposed to manually engineering structural and visual features.

Experiments on real data from Gmail corpus show that the structure does indeed contain useful information and that RiSER is able to exploit this information to outperform a strong baseline that only considers textual information. While we focus on emails, the techniques introduced in RiSER are generic and can be used for any document corpus containing richly formatted information such as HTML web pages and PDF documents.

We make the following key contributions in this paper:

- We identify that structural information in richly formatted HTML emails is a valuable signal that is ignored by existing classification approaches focused on textual content.

- We propose a 2-level neural architecture called RiSER that jointly learns to encode both the structure and the content of the email by examining the sequence of HTML tags that each text term is associated with.

- We demonstrate through experiments on two classification tasks on data from Gmail that RiSER learns enhanced email representations.

The rest of this paper is organized as follows: Section 3.2 provides more detailed background on the information available in the structure and markup, explains how we represent the input document, and lays out the problem definition for applications like information extraction. Section 3.3 introduces our two-level neural architecture that jointly encodes the HTML markup and the textual content of documents to learn a good representation. Section 4.5 reports on the performance of this architecture on two real-world classification tasks using data from Gmail. We compare the performance of this architecture with several baselines including ones that try to leverage manual feature-engineering to represent aspects of this information. The experimental results show that the models with the proposed architecture significantly increase recall at high precision compared to previous baselines by up to 8.6%. Section 3.5 summarizes areas of related work from the data mining, NLP, and ML research communities. Finally, Section 5.5 concludes the paper with a summary and potential directions for additional research.

3.2 Email Structure and Problem Setting

We focus on business-to-consumer (B2C) emails, the vast majority of which are machine-generated instantiations of predefined templates, accounting for up to 90% of all email traffic on the internet [52].

Figure 3.1 (*left*) depicts an example email from the author's inbox that contains a purchase confirmation. Unlike personal emails sent by humans, B2C emails like these often have rich HTML structure. This structure introduces a complex hierarchy which can be represented by a Document Object Model (DOM) tree [53], such as the one illustrated in Figure 3.1 (*right*). Each node in this tree represents an HTML tag in the message body, and its children consist of the HTML tags nested within it. The text of the message resides at the leaf nodes. Note that we have extended this DOM tree beyond the standard model to also include the subject of the email as a child node of the email root. A typical commercial email like the example in Figure 3.1 may contain around 2,000 leaf nodes. In fact, we have observed some messages, albeit few, that have as many as 10,000 leaf nodes and a depth of up to 200.

The use of rich HTML structure draws the attention of the human eye to different parts of the email. We find that the categorically similar emails often share similar structure. For example, bill reminder emails will format a “pay now” link close to the middle of the email, while hotel confirmation emails often contain tables of check-in and check-out information. This manifests as tables, headers, footers, highlighted and emboldened text in the markup. The key hypothesis we test in this paper is whether the markup in such richly formatted emails offers additional signal over just the textual content.

The remainder of this section describes how we convert this tree representation into a sequence of features that are suitable as input for the classification models described in the following section. The sequence consists of three types of features: *textual features*, *annotation features*, and *structural features*.

The textual features consist of a sequence of the first 200 terms (including punctuation) extracted from the DOM tree (including the subject branch) via in-order traversal. Limiting ourselves to 200 terms allows us to fully represent most emails without the risk of allowing very long emails to derail model training.

The annotation features indicate whether a particular text span contains an annotation. Several examples such as *Date*, *Price*, and *Time* are listed in Table 3.1. A term may correspond to one or more annotations. These annotations are extracted through a variety of methods including dictionary lookups and regular expression matching. Libraries of

Annotation Types	
Address	Location
Alphabet-Number	Price
Confirmation Number	Telephone Number
Date	Time
Establishment	Tracking Number

Table 3.1: Annotation features.

these annotators were developed over several years to support more traditional rule-based information extraction tasks. An entity detection library, for example, is also available as a Google Cloud API [54]. Readers can refer to Sheng et al. [48] for more detailed discussion on these annotators. In this work, we provide a way to incorporate the signals from these annotations.

Finally, we represent the structure of the document using XPaths [55]. An XPath is the sequence of nodes (HTML tags) extending from the root of the DOM tree to a leaf node. For example, the XPath of the term "ORDER" in the example email in Figure 3.1 is */html/body/table/tbody/tr/td/span/text*. All the terms in a paragraph (in a *p* node), for instance, might share the same sequence of HTML tags for their XPath feature.

We considered several alternative approaches to represent document structure. For example, one can identify the XPath patterns for text in key HTML tags like headers, tables, lists and then use these to mark each term. This approach is labor-intensive and more importantly ad-hoc and may not take into account patterns than an engineer has not visually inspected and considered important. For our problem, we are interested in an approach that can be applied to all the emails from various domains with different structural patterns without having to manually engineer structural features. Another possible representation is to use a visual blocks structure using external parsers [56]. Visual blocks are tree structures based on the visual layout of the page. Such an approach would depend on an external parser to produce features that can then be represented as

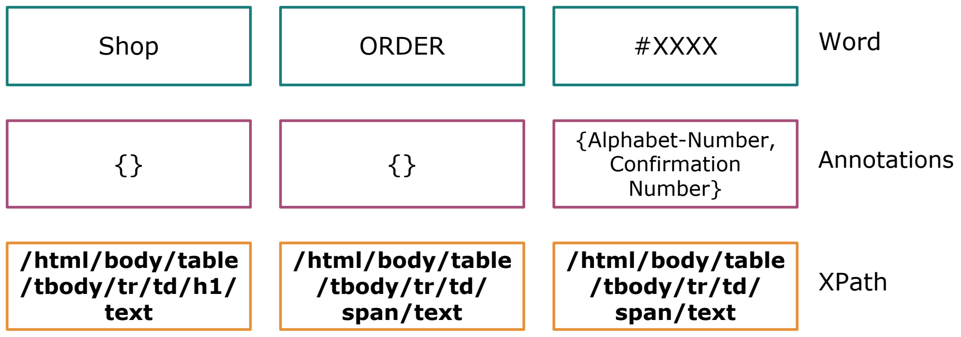

Shop
ORDER
#XXXX
Word
{}
{}
{Alphabet-Number, Confirmation Number}
Annotations
/html/body/table/tbody/tr/td/h1/text
/html/body/table/tbody/tr/td/span/text
/html/body/table/tbody/tr/td/span/text
XPath

Figure 3.2: Feature sets of three terms ”Shop”, ”ORDER” and ”#XXXX” from the top of the email in Figure 3.1.

part of the input example. We chose to use the XPath since it is simple, and provides a detailed representation of the markup. This allows us to directly test the hypothesis of whether the information in the markup is valuable for learning a better representation. We expect that using visual features from a sub-system that understands document layout could be interesting future work.

A complete set of features representing a single email document thus consists of a stream of up to 200 terms, the set of annotations that each term is a part of, and the XPath that each term resides at. Figure 3.2 denotes the features corresponding to three terms – ”Shop”, ”ORDER” and ”#XXXX” from the email in Figure 3.1. The terms ”Shop” and ”ORDER” do not contain any annotation, but ”#XXXX” has two annotations: *Confirmation Number* and *Alphabet-Number*. The bottom row contains the XPath in the DOM tree at which each of these terms appear.

We describe the architecture that combines these three features in Section 3.3 below, and explore the utility of different combinations of these features in Section 4.5.

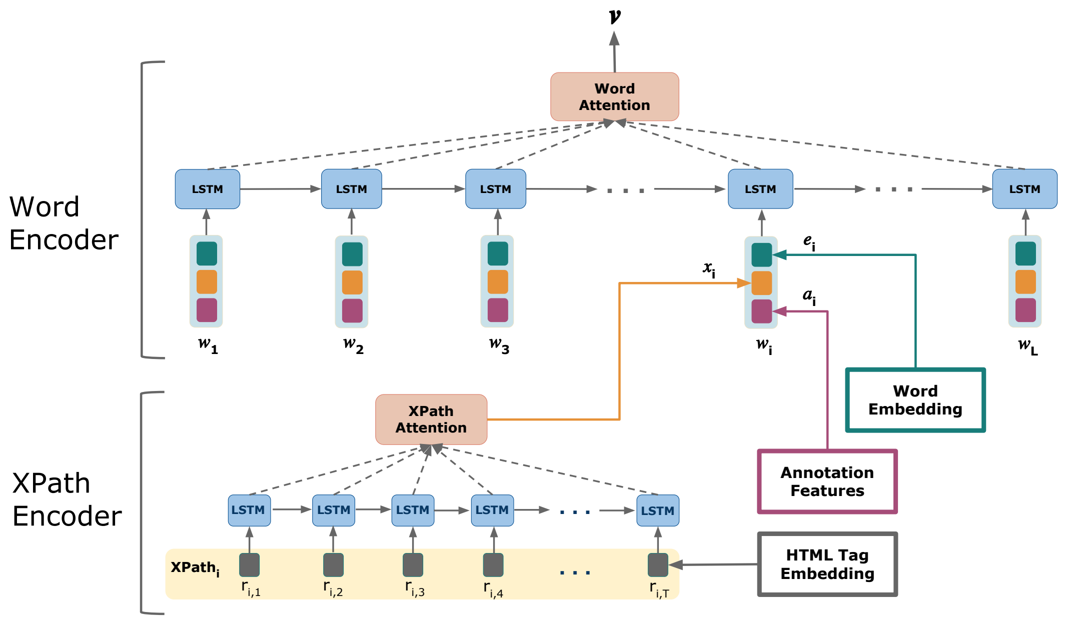


Figure 3.3: Overview of the RiSER architecture. The XPath Encoder iteratively encodes the tags in the XPath to each term in the email's DOM tree. Later, each XPath encoding (x_i) is combined with the corresponding word embedding (e_i) and annotation features (a_i) of the term to form a word representation (w_i). The Word Encoder then processes these word representations ($\{w_1, w_2, \cdots, w_L\}$) to learn an enhanced email representation (v). *(best viewed in color)*

3.3 Proposed Framework

In this section, we present our email representation framework – RiSER. The proposed model consists of two main components: an XPath encoder and a word encoder. At a high level, the XPath encoder models an email's DOM structure by encoding the tags along the XPath to a leaf node using an LSTM layer [57] while the word encoder combines word embeddings with the corresponding XPath encodings and additional features to learn an enhanced representation of the email. The enhanced email representation is then used as a feature for email classification. Figure 4.3 shows an overview of the model architecture. We describe the details of the framework and its components in the following sections.

3.3.1 XPath Encoder

Recall from Section 3.2 that an XPath is a sequence of HTML tags from root to leaf of an email's DOM tree. The proposed XPath encoder in Figure 4.3 summarizes this sequence with a vector of fixed length. In this work, we use an LSTM encoder with an additional attention mechanism [51] to represent the sequential elements (HTML tags), however, one is free to choose any standard sequence encoder to model the sequential input. More precisely, let the $XPath_i$ be a sequence of HTML tags with embeddings $[r_{i1}, \cdots, r_{iT}]$. In practice, we randomly initialize these embeddings and update them during training. We pass these embeddings through an LSTM layer to produce output vectors:

$$h_{it} = \overrightarrow{LSTM}(r_{it}), t \in [1, T] \tag{3.1}$$

Typical practices for extracting a final output representation from an LSTM layer include selecting the final output or averaging across all outputs. However, we observe that XPath sequences can be very long and highly repetitive. Furthermore, not all HTML tags may contribute equally to the email structure. For instance, HTML tags such as `strong`, `em` or `big` can be more informative than other much more common tags, such as `div`. Hence, we apply an attention mechanism [51] over the outputs of the LSTM layer in order to (1) extract such informative tags from long sequences and reward their contributions to the XPath representation, and (2) simultaneously reduce the impact of highly repetitive tags.

The final XPath encoding vector is computed as follows. First, we feed the LSTM output h_{it} through a fully connected layer with trainable weight matrix W_r and bias vector b_r, as well as a *tanh* activation function to produce the hidden vector u_{it}:

$$u_{it} = \tanh(W_r h_{it} + b_r) \tag{3.2}$$

Next we calculate an importance weight α_{it} for each HTML tag r_{it} in the sequence as a normalized similarity score between the hidden vector u_{it} and a trainable structure vector u_r:

$$\alpha_{it} = \frac{\exp(u_{it}^T u_r)}{\sum_i \exp(u_{it}^T u_r)} \tag{3.3}$$

Finally, we compute the *XPath encoding* vector x_i as a weighted sum of all the LSTM outputs:

$$x_i = \sum_t \alpha_{it} h_{it} \tag{3.4}$$

The proposed XPath encoder has the following two desirable properties. First, as we show in Section 3.4.5, the XPath encoding vectors capture information that is highly valuable for an email classifier. Second, the XPath encoder captures the relative hierarchy of an email's structure, i.e. two XPath sequences will have similar embeddings in the vector space if they correspond to closer leaf nodes in the DOM tree since they share long common subsequences. In the next section, we explain how the XPath encoder interacts with the word encoder.

3.3.2 Word Encoder

The word encoder is responsible for learning a rich representation of the email which not only summarizes the email's content but also encodes its HTML structure. In order to accomplish this goal, we go beyond the word-embeddings-based representations and capture additional features with structural information from the email. We create our rich word representation vector w_i by leveraging the following three components:

- *Word embeddings.* We construct our vocabulary with the most frequent 20,000 words in our data. Each word in the vocabulary is mapped to a word embedding vector through an embedding look-up matrix. Out-of-vocabulary (OOV) words are mapped to an *unknown* vector. In this work, we randomly initialize the word embeddings and jointly train them with the model. One can also use pre-trained word embeddings such as Word2Vec [58], Glove [59], or fastText [60].

- *XPath encodings.* The corresponding XPath of each word is passed through the XPath encoder to produce an XPath encoding. These XPath encodings implicitly capture the position of the word in the DOM tree. Since XPath sequences can be very long, we only keep the first T tags of the sequence and omit the rest. Note that T is a hyper-parameter in our framework.

- *Annotation vectors.* We represent the annotations a word corresponds to as a 10-D binary vector. Each dimension (0/1) indicates whether a word is annotated by a particular markup shown in Table 3.1. These features turn out to be very helpful, as we will show in Section 3.4.4.

To form improved word representations, the word embeddings, XPath encodings, and annotation vectors are concatenated for each term and fed into a fully connected layer with *tanh* non-linearity. Formally, given an email with L words, we represent each word in the sequence as:

$$w_i = \tanh(W_s[e_i, x_i, a_i]), i \in [1, L] \tag{3.5}$$

where $e_i \in \mathbb{R}^k$ is the word embedding vector, $x_i \in \mathbb{R}^l$ is the corresponding XPath encoding calculated in Equation (3.4), and a_i is the 10-dimensional binary vector representing the annotations mentioned in Section 3.2. $W_s \in \mathbb{R}^{m*m}$ is a trainable weight matrix, where $m = k + l + 10$.

With $[w_1, w_2, \cdots, w_L]$ we can now compute an email representation vector by using another LSTM layer to encode the sequence and apply an additional attention mechanism [51] over that layer:

$$h_i = \overrightarrow{LSTM}(w_i), i \in [1, L] \tag{3.6}$$

$$u_i = \tanh(W_w h_i + b_w) \tag{3.7}$$

$$\alpha_i = \frac{\exp(u_i^T u_w)}{\sum_i \exp(u_i^T u_w)} \tag{3.8}$$

$$v = \sum_i \alpha_i h_i \tag{3.9}$$

where v is the email representation vector that summarizes both the email structure and semantics.

3.3.3 Email Classification

We use the email representation computed above as a feature in email classification. More specifically, we use a linear layer to convert the email representation vector to a real-valued vector of size $|C|$, where C is the set of classes to predict. A final softmax layer is applied to obtain the normalized probabilities over the labels:

$$p = \text{softmax}(W_c v + b_c) \tag{3.10}$$

The training objective of our framework is cross-entropy loss:

$$loss = -\sum_{d \in D} \sum_{c \in C} p_c^g(d) \cdot log(p_c(d)) \tag{3.11}$$

where D is the set of training emails, C is the collection of email classes, $p_c(d)$ is the probability of predicting email d as class c, and $p_c^g(d)$ is a binary value indicating whether c is the correct label. The entire model is trained through back-propagation with respect to all parameters. Training details are further explained in Section 3.4.3.

3.4 Experiments

This section presents the results from several experiments on data from a large Gmail corpus to illustrate the effectiveness of the RiSER architecture. We focus on two binary classification tasks for this purpose. The first task is one of detecting if an email contains a bill. This model is used by the email service to determine if it wants to proactively remind the user when a bill is due. The second task is one of detecting if an email contains a hotel reservation confirmation. This model is also used to support intelligent applications such as travel planning. We refer to these two as the *Bill* and *Hotel* tasks. We pick these two tasks for illustrative purposes. The overall system that these classifiers are a part of includes several other classifiers (and extractors) for flights, calendar appointments, shipping confirmations, etc.

The experiments are designed to compare the performance of different RiSER variants (introduced in Section 3.4.2) against a strong baseline and to understand the incremental advantage of incorporating the annotations and XPath embeddings. Before describing the experimental setting, we first describe how the datasets are constructed including the source for the labels and sampling techniques used to deal with biases.

3.4.1 Dataset

The training and testing datasets are constructed by sampling from the Gmail corpus. During the course of this work, user privacy was protected through strict data access controls and data pre-processing to avoid training on sensitive data. Nobody involved with this project had access to visually inspect any of the data. Only terms matching the dictionary of top 20,000 terms are used, while the rest are replaced by an *unknown* identifier. Any text spans that contain potentially private data (such as addresses, dates, and phone numbers) are denoted by the corresponding annotation feature, and the underlying terms are replaced by the *unknown* identifier.

3.4.1.1 Labels

Ground truth labels for this dataset are derived from three sources: Microdata [61], manually defined parsers, and rule-based extractors. Microdata is a standard that enables senders to explicitly label and mark up their outgoing emails with structured information. For example, when sending a confirmation email to a customer, hotels can include markup to indicate that their emails are reservation confirmations, and even specify details such as confirmation numbers, addresses, check-in and check-out out dates and times, etc. While Microdata markup is precise, it is not widely adopted by all email senders, so the volume of annotations tends to be low.

Manually defined parsers are designed to extract category-specific fields from emails in lieu of Microdata. These are created on a per-sender basis and hand-crafted based on several instances of emails donated by users for this very purpose. For the purposes of email classification, we use the presence of a successful extraction from an email as a positive label for the category.

Note that we prioritize construction of parsers for high volume senders, since (a) parser construction is laborious, thus focusing on larger senders yields more labeled samples, and (b) the donated corpus is relatively small and thus contains a limited number of samples from low volume senders. Manually defined parsers tend to be highly precise but brittle, since small changes to the emails can often break the parsers, requiring human involvement and new donated emails to fix them.

Rule-based extractors consist of a manually engineered set of traditional information extraction techniques, such as dictionary lookups and regular expressions, that operate across senders. These are also constructed by leveraging the donated email corpus for both development and validation. These tend to have higher recall than the approaches above, but often at the cost of lower precision. Similar to the manually defined parsers, we use the presence of a successful extraction of an email as a positive label for the email.

3.4.1.2 Sampling

Despite drawing labeled examples from the three sources of ground truth mentioned above, the vast majority of emails remain unlabeled, and even among those that are labeled, the distribution is skewed towards higher volume senders. Thus, training classifiers using a uniformly random sample of data often leads to overfitting and poor generalization to smaller senders.

Domain-based sampling attenuates this issue by limiting the number of samples observed from high volume senders and boosting those from smaller senders, however this method can be too coarse for larger senders that have their own internal skew—e.g. the majority of amazon.com email might be classified as *purchase orders*, while a small fraction are *account memberships* or *bill reminders*.

We find a balance between these two methods by stratifying emails by the templates from which they are instantiated. This ensures that an even number of emails are represented from the *purchase order*, *account membership*, and *bill reminder* templates, along with emails from much smaller senders.

These templates are inferred through a one-time template generation process that clusters similarly structured emails into groups that are likely to have been instantiated from a single template. These techniques range from clustering on the sender and subject of an email [62] to clustering on the structure of the email body [63, 50]. In this work, we use structural clustering techniques based on a locality sensitive hash of the email body similar to the technique described in [48].

We split the data into train and test sets with a ratio of 9:1 while ensuring that samples belonging to the same sender domain appear in only one of these sets to prevent the effects of memorization. For the training data, we downsample the negative examples to yield a positive-to-negative ratio of about 1:100. We keep the original ratio in the test set. To put this into perspective, the resulting training sets have approximately *150M* total samples for the Bill task and *31M* samples for the Hotel task.

3.4.2 Experiment Configuration

We train and evaluate four variants of the RiSER architecture, each incorporating a different combination of the textual, annotation, and structural features described in Section 3.3.2.

RiSER-W uses the Word Encoder only and represents words using the *word embeddings* component only. That is, in Equation 3.5, the word representation w_i is represented by the word embedding e_i only.

RiSER-WA uses the Word Encoder and represents words with the *word embeddings* and *annotation features*.

RiSER-WX uses both the Word Encoder and the Xpath Encoder and represents words with the *word embeddings* and *XPath encodings*.

RiSER-WXA uses both the Word Encoder and the XPath Encoder and represents words with the *word embeddings*, *XPath encodings*, and *annotation features*. This variant is the one displayed in Figure 4.3.

The RiSER-W variant is simply an LSTM-based recurrent model with an attention mechanism, as it only includes word embeddings in its input layer. Recent studies in the literature have shown that attention-based LSTM models [51, 64] are excellent for modeling text documents and achieve state-of-the-art performance in document classification tasks by outperforming linear models, SVMs, and feed-forward neural networks. Therefore we consider RiSER-W as a baseline model to compare against the remaining RiSER variants.

Note that one can employ more complex neural architectures than just the RiSER-W architecture to improve email classification performance. However the focus of this work is to demonstrate the advantages of exploiting the rich formatting structure of emails through our proposed XPath Encoder and overall architecture. Moreover, we aim to demonstrate that textual content can be better utilized when combined with structural information for learning email representations.

We compare the performance of the RiSER-WA, RiSER-WX and RiSER-WXA variants against the baseline RiSER-W model to determine whether structural information and annotation features can improve the learned representations. We test this hypothesis by classifying emails into two semantic categories, *Bills* and *Hotels*. While we could have posed this as a multi-class classification problem, practical convenience of improving one model without affecting any other models motivated us to pose these as separate *binary classification* tasks. The general idea of leveraging structure to learn a better representation holds for multi-class, multi-label, or even regression tasks.

BILL				
Hyper-parameter	RiSER-W	RiSER-WA	RiSER-WX	RiSER-WXA
batch size	200	200	200	200
learning rate	0.001	0.0001	0.0001	0.001
optimizer type	Adam	Adam	Adam	Adam
word embedding dim	200	100	100	100
WE LSTM output dim	128	128	128	128
WE dropout rate	0	0.25	0.25	0
tag embedding dim	N/A	N/A	25	25
max XPath length	N/A	N/A	20	20
XE LSTM output dim	N/A	N/A	64	16
XE dropout rate	N/A	N/A	0.5	0.25

HOTEL				
Hyper-parameter	RiSER-W	RiSER-WA	RiSER-WX	RiSER-WXA
batch size	200	100	200	200
learning rate	0.001	0.0001	0.0001	0.001
optimizer type	Adam	Adam	Adam	Adam
word embedding dim	200	100	100	200
WE LSTM output dim	128	64	64	128
WE dropout rate	0.5	0.25	0.25	0.25
tag embedding dim	N/A	N/A	50	25
max XPath length	N/A	N/A	20	30
XE LSTM output dim	N/A	N/A	16	64
XE dropout rate	N/A	N/A	0.5	0.25

Table 3.2: Selected hyper-parameters for the RiSER variants for the Bill and Hotel classification tasks. WE stands for Word Encoder and XE stands for XPath Encoder.

3.4.3 Hyper-parameters and Training

For all four variants, we perform grid search over the word embedding dimension ({50, 100, 200}), the word encoder LSTM output (hidden state) dimension ({64, 128, 256}), the initial learning rate ({0.01, 0.001, 0.0001}), the batch size ({50, 100, 200}), the word encoder dropout rate ({0, 0.25, 0.5}), and the optimizer type (AdaGrad [65] and Adam [66]). For the Adam optimizer, we use the default settings suggested by the authors ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$).

For the RiSER-WX and RiSER-WXA models we also search over the following additional hyper-parameters: the HTML tag embedding dimension ({25, 50, 100}), the XPath encoder LSTM output dimension ({16, 32, 64}), the XPath encoder dropout rate ({0, 0.25, 0.5}), and the maximum XPath length ({20, 30, 40}).

We use the norm clipping trick with a *threshold* of 5.0 in all four models to avoid the exploding gradient problem.

For each model, we use the Vizier [67] platform to select the best combination of hyper-parameters based on AUC-PR after a maximum of 1M training steps. The selected hyper-parameters for each model are shown in Table 3.2. We later continue training the selected models until they converge and report their highest numbers on the test data in the next section.

3.4.4 Results and Discussion

Classifier performance is often measured using area under the receiver operating characteristic curve (AUC-ROC). However, when dealing in datasets with high class skew—which is often the case for email—the area under the precision-recall curve (AUC-PR) provides a much fairer representation of model performance. In practice, even the AUC-PR metric is inadequate given that applications for email generally require extremely high precision. For example, misclassifying an important email as spam is an incredibly bad user experience.

Thus, we evaluate our models based on their *recall at a fixed level of high precision*. By requiring that all models reach a prespecified precision threshold, we minimize bad user experiences. Fixing the precision then allows us to evaluate different models by the extent of their coverage at that level of precision. That being said, we also report the

BILL				
Model	R@P=0.85	R@P=0.90	R@P=0.95	AUC-PR
RiSER-W	76.4	71.5	57.4	85.4
RiSER-WA	76.8	73.0	63.1	85.9
RiSER-WX	76.6	73.5	64.1	86.2
RiSER-WXA	78.0	73.7	66.0	85.6

HOTEL				
Model	R@P=0.85	R@P=0.90	R@P=0.95	AUC-PR
RiSER-W	95.9	95.0	92.4	97.1
RiSER-WA	97.3	95.7	93.2	98.0
RiSER-WX	96.3	95.5	93.5	97.2
RiSER-WXA	97.4	96.3	92.9	97.7

Table 3.3: Recall at a fixed level of precision and AUC-PR performance metrics for four RiSER variants trained for the Bill and Hotel classification tasks.

AUC-PR metric for clarity. The experimental results on both datasets are summarized in Table 3.3.

3.4.4.1 Recall-at-fixed-precision (R@P)

Results show that our RiSER-WXA variant overall gives the best performance across both datasets. The improvement in performance depends on the data type. For more complex dataset, such as Bill, the RiSER-WXA variant outperforms the baseline model (RiSER-W) by 2.2% on R@P(0.9), and by 8.6% on R@P(0.95) metrics. Note that recall tends to decrease drastically for the baseline model as the precision threshold increases, while RiSER-WXA demonstrates more robust performance. This leads to a larger gap between both models' recall at higher precision thresholds. On the other hand, improvements in the Hotel task are smaller compared to the Bill task, leading to 1.3% on R@P(0.9), and by 0.5% on R@P(0.95) metrics. We also report the experimental results of RiSER-WA and RiSER-WX variants in order to better evaluate the effects of each component.

Effect of annotation features. Our experiments show that using crafted annotation features on top of word embeddings provides important performance gains over the baseline. With respect to the R@P(0.95) metric, the RiSER-WA variant outperforms the RiSER-W variant by 5.7% on the Bill task, and 0.8% on the Hotel task. Similarly, the gap between recall is smaller (1.5% and 0.7%) as we decrease the precision threshold.

We believe the reason behind the performance boost that comes with the annotation features is that they allow us to recover meanings in words that cannot be captured with the defined vocabulary. To be more specific, emails include dates, numbers, addresses such that most of these terms are anonymized due to privacy constraints and are embedded with the *unknown* vector. Thus, including the annotation features allows our framework to capture these additional signals, which intuitively improves the model's ability to correctly classify an email. As a toy example, an email without a date is most likely not a hotel reservation.

Effect of email structure. Combining XPath encodings with word embeddings leads to an improvement of 6.7% in the Bill task, and 1.1% in the Hotel task with respect to the R@P(0.95) metric. More importantly, by comparing the performance of RiSER-W and RiSER-WX, we verify that incorporating the *HTML structure* into the learning process leads to *improved email representations.* In Section 3.4.5, we illustrate this with some example emails in which the inclusion of HTML structure results in better classification.

While improvements of the order of 5% to 10% may seem small, they represent a significant reduction in "bad experiences" for users of Gmail. Consider, for example, a service that extracts structured data from these emails to remind people when their bills are due or answers questions about their hotel reservations. The improved models directly translate to improved recall for both these assistive experiences.

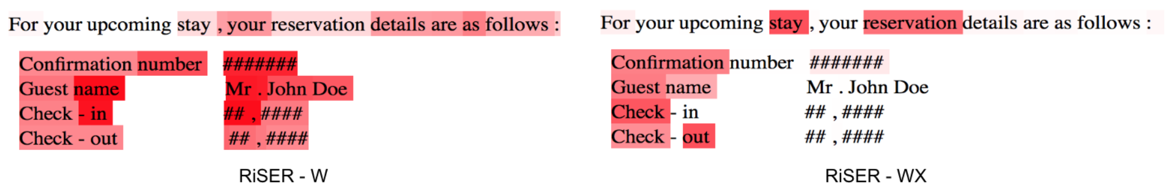

For your upcoming stay , your reservation details are as follows :
Confirmation number #######
Guest name Mr . John Doe
Check - in ## , ####
Check - out ## , ####
RiSER - W
For your upcoming stay , your reservation details are as follows :
Confirmation number #######
Guest name Mr . John Doe
Check - in ## , ####
Check - out ## , ####
RiSER - WX

Figure 3.4: Word attention weights on an example donated Hotel email using the RiSER-W and RiSER-WX variants. Brighter red highlighting indicates a higher attention weight for the corresponding term relative to the other terms in the email.

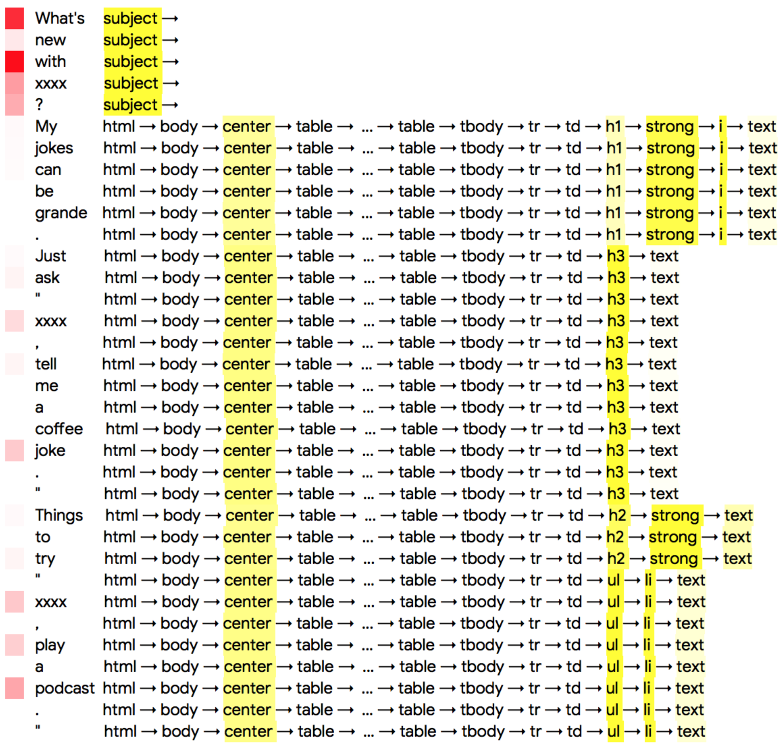
3.4.4.2 AUC-PR

The AUC-PR metric generally improves for RiSER-WA, RiSER-WX and RiSER-WXA variants compared to the baseline variant on both classification tasks. However, the improvements are not always in line with the improvements to the recall-at-fixed-precision metric. Surprisingly, unlike the R@P metric, the RiSER-WXA variant does not have the highest AUC-PR for either task. For the Bill task, the RiSER-WX variant performs the best (86.2%), while for the Hotel task, the RiSER-WA variant has the highest AUC-PR (98.0%).

3.4.5 Qualitative Analysis

In order to get an intuition into how RiSER utilizes the HTML structure of an email, we visualize and compare the attention weights of two architecture variants: one that excludes the XPath encodings and one that includes the XPath encodings. Here, we use the RiSER-W and RiSER-WX variants trained for the Hotel classification task. We illustrate the resulting attention weights of the two models applied to a single donated email example in Figure 3.4. Note that we obfuscate key information for privacy reasons.

In comparing the term attention weights of the RiSER-W and RiSER-WX models, we find that the latter is much better at attending to terms that are semantically relevant to the hotel reservation class. For example, terms such as ‘`stay`’, ‘`reservation`’,



What's	subject →
new	subject →
with	subject →
xxxx	subject →
?	subject →
My	html → body → center → table → ... → table → tbody → tr → td → h1 → strong → i → text
jokes	html → body → center → table → ... → table → tbody → tr → td → h1 → strong → i → text
can	html → body → center → table → ... → table → tbody → tr → td → h1 → strong → i → text
be	html → body → center → table → ... → table → tbody → tr → td → h1 → strong → i → text
grande	html → body → center → table → ... → table → tbody → tr → td → h1 → strong → i → text
.	html → body → center → table → ... → table → tbody → tr → td → h1 → strong → i → text
Just	html → body → center → table → ... → table → tbody → tr → td → h3 → text
ask	html → body → center → table → ... → table → tbody → tr → td → h3 → text
"	html → body → center → table → ... → table → tbody → tr → td → h3 → text
xxxx	html → body → center → table → ... → table → tbody → tr → td → h3 → text
,	html → body → center → table → ... → table → tbody → tr → td → h3 → text
tell	html → body → center → table → ... → table → tbody → tr → td → h3 → text
me	html → body → center → table → ... → table → tbody → tr → td → h3 → text
a	html → body → center → table → ... → table → tbody → tr → td → h3 → text
coffee	html → body → center → table → ... → table → tbody → tr → td → h3 → text
joke	html → body → center → table → ... → table → tbody → tr → td → h3 → text
.	html → body → center → table → ... → table → tbody → tr → td → h3 → text
"	html → body → center → table → ... → table → tbody → tr → td → h3 → text
Things	html → body → center → table → ... → table → tbody → tr → td → h2 → strong → text
to	html → body → center → table → ... → table → tbody → tr → td → h2 → strong → text
try	html → body → center → table → ... → table → tbody → tr → td → h2 → strong → text
"	html → body → center → table → ... → table → tbody → tr → td → ul → li → text
xxxx	html → body → center → table → ... → table → tbody → tr → td → ul → li → text
,	html → body → center → table → ... → table → tbody → tr → td → ul → li → text
play	html → body → center → table → ... → table → tbody → tr → td → ul → li → text
a	html → body → center → table → ... → table → tbody → tr → td → ul → li → text
podcast	html → body → center → table → ... → table → tbody → tr → td → ul → li → text
.	html → body → center → table → ... → table → tbody → tr → td → ul → li → text
"	html → body → center → table → ... → table → tbody → tr → td → ul → li → text

Figure 3.5: Term and XPath attention weights. Red highlighting indicates the term attention weights. Yellow highlighting indicates the XPath's HTML tag attention weights (row normalized). Brighter colors indicate higher attention weights.

'`confirmation`', and '`guest`' are attended to while the RiSER-W model attends to nearly all words. In fact, it attends more to those that are less semantically relevant in this example.

To get an intuition as to why structural information helps these models attend differently to these terms, we inspect the attention weights of the XPath tags in addition to the term attention weights. We visualize these weights for a donated email example in Figure 3.5. Red highlighting indicates term attention weights while yellow highlighting indicates the XPath's HTML tag attention weights.

Our first observation is that the use of structure helps the model understand the boundaries of different parts of the email. In this particular example the model attends to terms that belong to the subject more than the remainder of the text stream. Note that the word attention weights drop suddenly at the boundary of the subject and the beginning of the message body. We observed this pattern across many donated examples.

Our second observation is that the model also attends more to HTML tags that are often used to highlight important text. In this example, the model pays particular attention to meaningful tags such as `strong` (which indicates if a word is bold-ed), `h1`/`h2`/`h3` (which indicate headers), and `ul` and `li` (which indicate list items). These types of markup are generally used to draw the user's attention to specific text in a rendered email, often containing important information. Similarly, note that the `center` tag, which aligns all inner HTML to the center of the display, is also highly attended to.

In summary, from inspecting the attention weights of the RiSER architecture with and without XPath encodings, we observe that including structural information improves attention at the term level by helping the model learn about boundaries in documents (e.g. subject vs. message body), as well as learn to focus on parts that the original sender intended to draw the user's attention to (e.g. `strong`, `center`, etc.).

3.5 Related Work

Our work builds on contributions from several research areas including natural language processing, data mining, web mining, and deep learning. Here we discuss related work in two main categories: text representation and email representation.

There is a rich history of work in text representation [68, 69] for various tasks like sentiment classification [70] (e.g. positive or negative reviews) and genre detection (e.g. sports, news, entertainment, etc.). More recently, the availability of large amounts of text data has led to unsupervised approaches for learning a good distributed representation for words. Techniques like Word2Vec [71] and Glove [59] have produced pre-trained embeddings for words that can be combined with more complex neural approaches.

The success of these approaches are later combined with the power of recurrent neural networks (RNNs) which led others to learn distributed representations of a larger piece of text like sentences [72], paragraphs [73], and documents [74]. Several variants of RNNs like convolutional-gated RNN [75], tree-LSTM [76] and Quasi-RNN [77] are proposed as alternative architectures. Apart from recurrent neural networks, methods employing convolutional networks [78, 79] have also produced compelling results for text classification tasks.

Recently, authors in [51] have argued that the hierarchical structure of documents helps to learn better representations. Acknowledging that the importance of words or sentences may vary depending on the context, they use an attention mechanism [80] while hierarchically encoding words into sentences and sentences into documents. In addition to the hierarchical structure, discourse structure [81] in the text, which represents the linguistic organization of a text as a tree, has also been studied in recent document representation work [82, 83, 64].

Each of these lines of research are based solely on the textual content assuming that the documents are plain-text. In this paper, we focus on machine-generated emails which have a rich HTML structure. In addition to leveraging the textual context, we encode the structural markup to learn a better representation. Note that in our work, the term "structure" corresponds to a formatting structure of an email, rather than the linguistic structure mentioned above. Combining our framework with the aforementioned architectures is possible and makes for promising future work.

Learning good email representations has been an area of interest for multiple research groups [49, 48, 50]. In addition to information extraction, email classifiers are useful for other applications like automatic foldering [84, 46], spam classification [85], and message priority ranking [86].

Lastly, systems like Fonduer [87] tackle richly formatted documents for the task of

Knowledge Base Construction (KBC). Fonduer shows that augmenting textual features with structural and visual features produced by a library that parses the HTML markup and providing it to a strong baseline like a bi-directional LSTM with attention can indeed provide a significant improvement over just using the text features. Our results agree with the findings in Fonduer – formatting information does indeed help us learn a better representation for an email. In contrast to the approach of engineering a set of structural features (examples from Fonduer include HTML tag of the term, HTML tag of the parent, tabular features like row number, column number, n-grams from all the cells in the same row) our approach simply relies on learning an encoding using the sequence of HTML tags that appear in the path from the root to the node containing the term. Our results show that this is a powerful approach when combined with well-known building blocks like LSTMs and an attention mechanism. While we do not implement the nearly 40 structural, tabular, and visual features implemented in Fonduer, an interesting piece of future work is to study if XPath encodings as learned by our model can be as effective as the sophisticated features manually engineered in Fonduer.

3.6 Conclusion

In this paper, we study the problem of learning representations for richly structured emails. Considering the fact that most of today's emails are machine generated, we argue the rich formatting used in these emails contains highly valuable information that can be used to learn better representations. To address this gap in the literature, we proposed a novel framework called RiSER. Our framework combines three components from the email in order to learn an enhanced email representation: 1) textual content, 2) HTML structure, 3) manual annotation features. We show the effectiveness of our approach by evaluating the individual components as well as the full framework on email classification task across two datasets from a large email service. The experimental results and visualizations further indicate that our framework is learning an improved representation for emails and is capable of capturing valuable information buried in the email's HTML structure.

We plan to explore multiple avenues of future research. We expect that leveraging structure will be valuable for other challenging tasks like information extraction from emails and web pages. We are considering ways to extend the RiSER framework to other types of documents with rich structure. For instance, PDFs and scanned images contain many informative layout and structural signals. However such documents do not contain XPath-style markup. We are also exploring alternative ways to represent the layout and formatting information from such documents.

Chapter 4

Multi-Resolution Attention for Personalized Item Search

4.1 Introduction

Recent years have seen rapid growth in popularity and complexity of online e-commerce and content-sharing platforms (*e.g.* music streaming [88], video streaming [89], photo sharing [90]). Consequently, developing a high-quality search engine has become one of the key objectives of these online platforms with millions of active users. Despite their contextual differences, all of these platforms bear the common challenge of retrieving suitable contents from a large searchable database to satisfy their users' search intents.

Users interact with such platforms in highly personalized ways [91]. The same search query entered by different users is likely to carry different search intentions, due to the diverse nature of personal taste and preferences [92, 93]. To that extent, historical interactions of users serve as a great asset for the problem of personalized item search to improve users' search experiences. For example, one can look for *relevant* signals within the user history that can inform users' intent behind their search query. The idea of utilizing users' history to better understand their search needs has been widely studied in the literature and proven valuable for various domains including product search [94, 95, 96], web search [97, 98], microblog search [99], and video search [100, 101].

Researchers have explored various directions to model user history, most of which are naturally formulated as a sequence modeling problem, since user history data often originate as a sequence of (ordered) interactions (e.g. purchases, watches, likes). Amongst the previously proposed mechanisms, self-attention [102] has gradually become a key component in sequence modeling tasks, leading to state-of-the-art results across many domains, including natural language processing [103], speech recognition [104], and recommender systems [105]. The self-attention mechanism has also proven useful in personalized item search, thanks to its ability to detect attention weights from the input event sequence with respect to the given context (in this case, search query). Such attention weight distribution intrinsically carries a notion of relevance between the search query and user history, leading to contextualized personalization.

Most of the self-attention based and other sequential models by design account for sequential signals rather than temporal signals. However, the latter aspect has significant implications for personalized item search. Since the user interactions take place at aperiodic points in time [106], there can be gaps between the sequential patterns and temporal patterns of user behaviors (as illustrated in Figure 4.2). This entangles different explanatory factors unique to personalized item search. Intrinsically, the time spans between the search query and user history items directly affect their degree of relevance (e.g., interaction occurred a day ago vs. a month ago). While there are emerging efforts to incorporate temporal information into neural sequential recommendation models [107, 108, 109, 110, 111], where the goal is to recommend items that are likely to be of interest to users solely based on their past interactions (without the existence of a search query), this research direction has not been adequately studied for the setting of personalized item search.

In this work, we propose *multi-resolution attention* for personalized item search. Multi-resolution attention effectively retrieves relevant items for users while accounting for higher-order temporal dependencies between their search query and item history. The key innovative idea behind our method is to compute the relevance between the search query and the history items over various temporal regions (or subspaces), which in turn can recognize and incorporate users' interests from various temporal resolutions. We achieve this by a novel multi-head attention formulation that explicitly enforces different attention heads to cover parts of the sequence that belong to distinct temporal regions with adaptive time boundaries, which are also learned jointly with the rest of the model. Our approach comes in two variants, each designed to accommodate different temporal densities of real-world data.

We evaluate the proposed approach on a public benchmark dataset and compare it with strong baseline approaches, including the adaptations of two state-of-the-art temporal models [109, 111] originally proposed for the sequential recommendation task. Our experiments showcase that multi-resolution attention consistently achieves superior performance across five different domains, outperforming the best baseline by up to 4.7%. Moreover, our method is a parameter efficient alternative to existing embedding-based [109] and kernel-based [111] methods, providing a new perspective on modeling the complex temporal nature of user history.

By design, our method operates on a space of discrete identifiers of items, circumventing the assumption that additional contextual information is available. Recent works have shown that such information may not be available nor feasible to obtain for many settings such as content-sharing platforms [112], while their usefulness is also in debates [113]. Therefore, we focus on the orthogonal problem (that is, how to build a model on top of these embeddings to account for temporal dependencies in data), which can potentially extend to multiple domains with different contextual flavors (*e.g.*, textual descriptions, images, and other metadata features [114, 115]).

4.2 Related Work

Personalized item search is a generic concept aiming to improve users' search experience by retrieving personalized items from a large searchable database. It is important to note that, while a search query might possess different forms (*e.g.*, text, image, voice), our work particularly focuses on textual queries. Conceivably, the most popular application domain for personalized item search is online e-commerce, wherein the term product is generally used as a substitute for the term item. Studies on personalized *product search* [94, 95, 96, 116, 117, 118, 119] essentially aim to link search queries with products (often via their contextual information) while taking into account the users' previous action logs within the platform.

Being the earliest study to investigate personalization in product search, authors in [94] proposed a hierarchical embedding model to learn latent semantic representations of users, products and queries with their associated language data (e.g. review), and retrieve products according to the similarities directly measured in this latent space. Similarly, authors in [119] employed a review-based transformer model to encode user, product and query relations. In another study [95], authors proposed a technique consisting of two attention networks, each designed to independently capture short and long-term user interests. However, their method defines the short/long-term interest solely based on the sequential order of interactions (*e.g.*, short-term means last m interactions), which can not capture the rich temporal patterns found in data. Considering that the previous studies often model user history and their search query as separate signals, authors in [96] more recently argued that the two signals are tightly connected, and investigated the potential of personalization with respect to query characteristics. While their findings highlight the need for query-aware personalization in product search, their method essentially treats all previous interactions as a set, ignoring their order, let alone temporal dynamics.

The pursuit of personalization has also become one of the main pillars in the development of search engines for content-sharing platforms [120, 112]. The diverse contextual nature of such platforms imposes unique challenges to learn meaningful representations of users and items. In the lack of descriptive information, authors in [112] combined the discrete user history signal with the corresponding provider information to perform personalized item retrieval. Another work [120] developed a method to jointly learn user and listing embeddings for personalized home listing search, utilizing multiple in-session (clicks, host contacts) and meta-data signals.

Despite their great success, the aforementioned studies for personalized item search do not leverage the rich temporal signals found in data. Being the first to address this problem, our work aims to recognize and capture higher-order temporal dependencies between users' search queries and item history using a novel multi-resolution attention mechanism.

Sequential recommendation is another line of related work to ours, with a prominent difference that the users can not specify their information needs explicitly. Therefore, the goal is to recommend items that are likely to be of interest to users solely based on their history. Amongst the pioneer studies, authors in [105] leveraged the self-attention mechanism for adaptive summarization of user history. Memory networks are adopted in [121] to memorize the anchor items that drive future user actions. Another study [122] proposed a hierarchical gating network to adaptively control which latent features of items will contribute to the downstream task.

There have been recent studies to model the *temporal aspect* of user history in the context of sequential recommendation [107, 108, 109, 110, 111]. Amongst these studies, authors in [109] proposed a time-aware self-attention module, which—in addition to relative position representations [123]— learns relative time interval representations to jointly capture both the sequential and the temporal nature of user interactions. The idea of modeling time intervals between user interactions is also studied in a recent work [111], wherein the authors instead employed a combination of different time kernels to calibrate the attention weights between user interactions based on their relative time intervals, providing a parameter efficient alternative to the embedding based approach proposed in [109].

These studies fundamentally differ from the setting of personalized item search because the presence of a search query marks a pivotal point in time, which has distinct indications in modeling the temporal signals with respect to user history. Needless to say, different search queries entered by users have varying temporal relations to their interaction history, the extent of which also depends on the context of the query (*e.g.* "sports watch" vs "Fitbit Versa 3"). Nonetheless, our experiments include adaptations of the aforementioned temporal recommendation models [109, 111] as baselines to (1) better assess the efficacy of our proposed approach, and (2) bridge the methodological gap between the two problem settings.

4.3 Temporal Resolution of User Interactions

We first provide insights into the complex temporal dynamics of user interactions observed in a real-world setting [106] and further draw connections to key motivations behind our proposed method.

In the context of personalized item search, there exist numerous factors contributing to the relationship between a search query and the recorded user interactions (*i.e.*, history). Such relationships are often tightly connected to the temporal aspect of the user histories

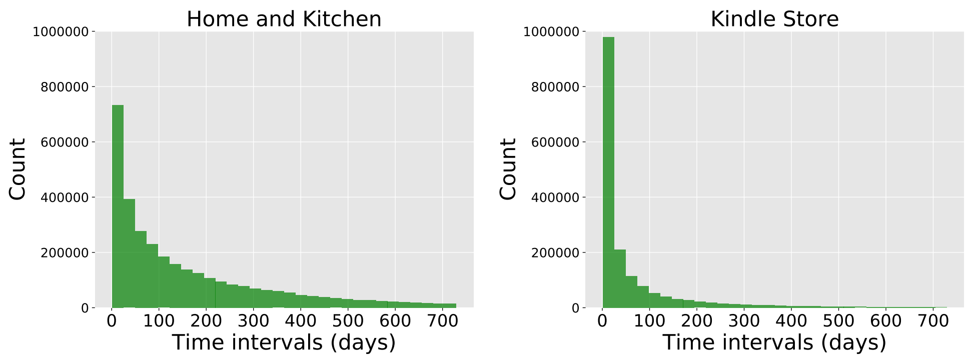


Figure 4.1: Histogram of time intervals between consecutive interactions of users.

[124, 125, 109, 126], as well as the users' search intents [96] (which, in turn, are reflected on their query formulations [92, 127]). For instance, while some search queries (e.g. "action movies") might exhibit longer-term dependencies on user histories, some others (e.g. "humidifier") are triggered by users' short-term interests, hence might not depend on their longer-term history. Moreover, such dependencies might display strong periodic patterns, as in the case of recurrent purchases of grocery products or cleaning supplies.

To study the complex and non-linear nature of these temporal dependencies in data, one needs to examine the temporal dynamics of user interactions. Figure 4.1 plots the histogram of time intervals between two consecutive interactions (of the same user) for two broad product categories, showing that the users' interaction patterns vary depending on the context of their search. Furthermore, to observe users' interaction patterns more closely for both categories, we also plot the most recent interactions of the same set of five active users, together with the temporal information of their interactions. Figure 4.2 shows that even the same user might exhibit vastly different temporal patterns when interacting with different categories (*e.g.*, users 3 and 4), revealing one of the major challenges in personalized item search. We also observe that user interactions tend to be concentrated (or grouped) over multiple temporal spans, each corresponding to a relatively narrow time period (e.g. 2-3 days). On the other hand, the gaps between such grouped

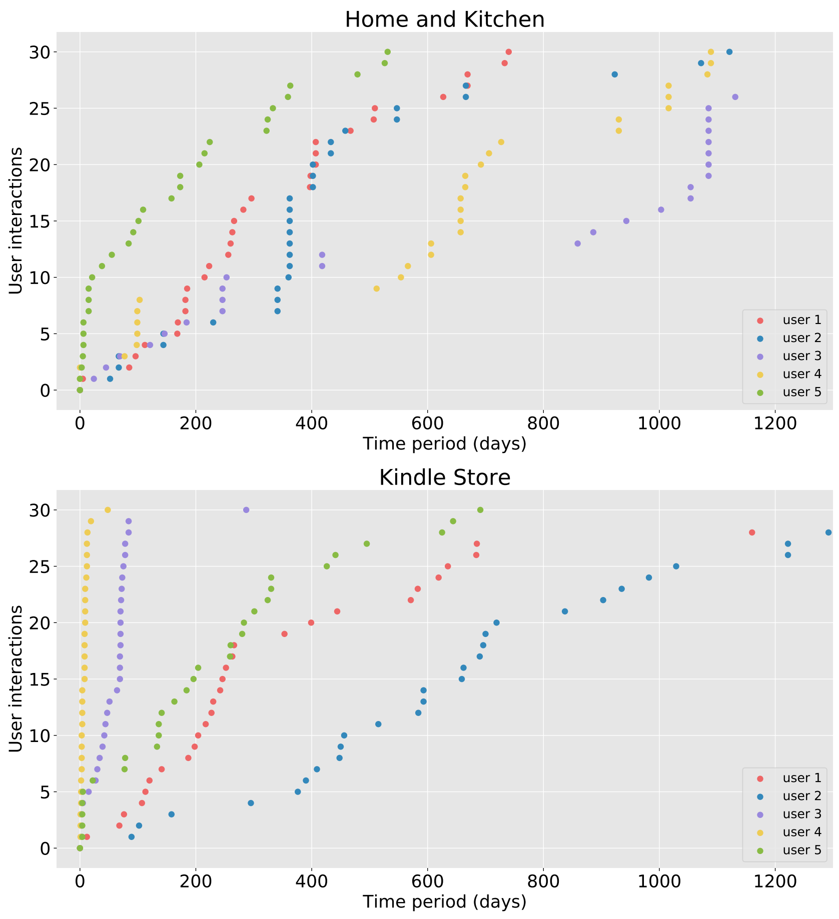


Figure 4.2: Last thirty interactions of the same set of five users who actively interact with both categories. Y axis represents the users' interactions in chronological order, while the X axis represents the time period (in days) with respect to their latest interaction.

interactions can be very large (e.g., over a year). For the remaining of this paper, we refer to such grouped temporal patterns as the ***temporal resolutions*** of user interactions.

We further argue that the potential for personalization differs across these resolutions. With that being our key intuition, we propose a novel approach (*multi-resolution attention*) that can (i) adaptively recognize such temporal resolutions and (ii) accurately model the diverse dependency patterns between search queries and user histories across these resolutions. The next section explains the details of our method, which is illustrated in Figure 4.3.

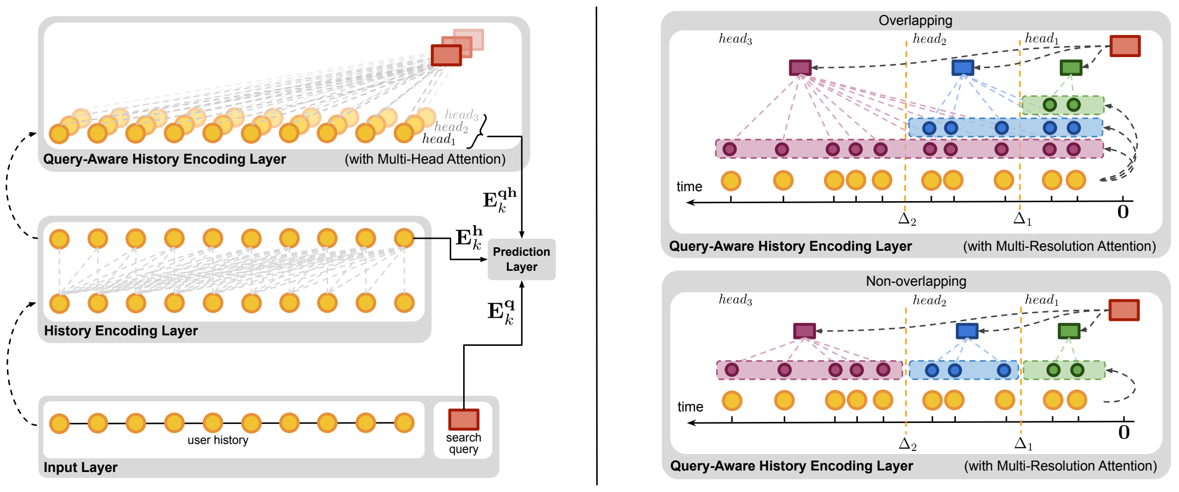

Query-Aware History Encoding Layer
(with Multi-Head Attention)
$head_1$
$head_2$
$head_3$
$\mathbf{E}_k^{\mathbf{qh}}$
$\mathbf{E}_k^{\mathbf{h}}$
$\mathbf{E}_k^{\mathbf{q}}$
Prediction Layer
History Encoding Layer
user history
search query
Input Layer
Overlapping
Non-overlapping
time
Δ_2
Δ_1
0
Query-Aware History Encoding Layer
(with Multi-Resolution Attention)

Figure 4.3: An illustration of our proposed method architecture. The left side shows the high-level components of our method, while the right part shows the proposed *multi-resolution attention* variants: *overlapping* (top) and *non-overlapping* (bottom).

4.4 Proposed Method

4.4.1 Problem Setting

We start with formally introducing our problem and the notations used in this paper. Let $\mathcal{I}$ denote the set of items, $\mathcal{U}$ denote the set of users. For each user $u \in \mathcal{U}$, we are given the following inputs: *(i)* a recent search query q^u, *(ii)* a time-ordered list of previously interacted items $S^u = (v_1^u, \cdots, v_{|S^u|}^u)$ where $v_i^u \in \mathcal{I}$, and *(iii)* a list of timestamps $T^u = (t_1^u, \cdots, t_{|T^u|}^u)$ corresponding to each interaction, where $t_1^u \leq \cdots \leq t_{|S^u|}^u \leq t_{q^u}$, with t_{q^u} being the timestamp of q^u, and $|S^u|$ (or $|T^u|$) denoting the number of interactions the user u previously had with the system. Our goal is to predict the $(|S^u|+1)$th item that the user u will interact with.

The main notations used in our paper are summarized in Table 4.1.

Notation	Description
$\mathcal{I}$,$\mathcal{U}$	item and user set
Q^u	user u's query sequence (in chronological order)
S^u	user u's item sequence (corresp. to Q^u)
T^u	user u's timestamp sequence (corresp. to Q^u, S^u)
N	maximum sequence length
d, d_q	latent dimensions
h	number of attention heads
$\mathbf{q}, \mathbf{w}$	query and word embedding vectors, respectively
$\mathbf{M}, \mathbf{P}$	item and position embedding matrices, respectively
$\mathbf{T}$	relative time interval matrix
$\widehat{\mathbf{E}}$	input embeddings (corresp. to S^u)
$\mathbf{E}^{(l)}$	input embeddings after (l)th self-attention block
$\mathbf{E}^{\mathbf{q}}$	query embeddings (corresp. to Q^u)
$\mathbf{E}^{\mathbf{h}}$	input history encoding
$\mathbf{E}^{\mathbf{qh}}$	query-aware history encoding
$\mathbf{H}$	output representation of the model
Δ_i	time boundary for ith attention head

Table 4.1: Main notations used in this paper.

4.4.2 Preliminaries

This section introduces preliminary techniques that serve as the building blocks of our framework, which are explained in the context of recommender systems to maintain contextual consistency.

The **self-attention mechanism** [102] aims to capture the importance (attention) weights of the sequential inputs (in our case, items) that are identified through the inner products of item representations. The items with higher attention weights have more contributions to the final output representation, and consequently, to the final downstream task. Such mechanism in principle assumes that the output of a given sequential input is relevant to only part of the sequence, which makes it a natural and desired instrument for recommendation tasks [105, 121, 96, 128, 109]. It is formally defined as:

$$\text{Attention}(Q, K, V) = \text{Softmax}(\frac{QK^\top}{\sqrt{d_{attn}}})V \tag{4.1}$$

where Q, K, and V respectively denote the queries, keys, and values of items in the sequence. Here it is important to note that the above term “query” is domain-agnostic, *i.e.*, it is not tied to the notion of “search query” in the context of our work. This mechanism relies on the positional embeddings to recognize and capture the sequential order of items. Hence, in applications, the vector representation for each position is combined with the corresponding item embeddings.

Causality. Due to the nature of our problem, the model should only take into account the previous items when predicting the next item. Therefore, we need to prevent leftward information flow (leak) in self-attention computation. This is achieved by masking the upper triangular entries of $QK^\top$, that is, $(QK^\top)_{i,j} = -\infty \; \forall i < j$.

The **self-attention block (SAB)** is defined as a combination of self-attention and point-wise feed-forward network (FFN) layers:

$$\text{SAB}(X) = \text{FFN}(\text{Attention}(XW^Q, XW^K, XW^V)) \tag{4.2}$$

where $W^Q \in \mathbb{R}^{d \times d_{attn}}, W^K \in \mathbb{R}^{d \times d_{attn}}, W^V \in \mathbb{R}^{d \times d}$ are the (linear) projection weight matrices. FFN is essentially a two-layer MLP with ReLU activation, applied *independently* to each position of the input.

4.4.3 Query-Aware Personalization

This section focuses on building the base of our architecture by leveraging several neural components introduced in Section 4.4.2. The subsequent section is dedicated to the novel components of our method, explaining how we leverage the temporal signal.

For training purposes, we transform the input sequence of each user into a fixed-length sequence of N interactions $S^u{=}(v^u_1, \cdots, v^u_N)$. If the input sequence has more than N items, we only consider the most recent N interactions and omit the remaining items, while if the sequence length is less than N, we left pad the sequence until it reaches the length N. Same procedure is also applied to the corresponding sequence of timestamps $T_u = (t_1^u, \cdots, t_N^u)$ and the sequence of queries $Q^u = (q_1^u, \cdots, q_N^u)$. Each query q_i^u results in interaction with item v_i^u. For simplicity, we assume that both q_i^u and v_i^u have the same timestamp (t_i^u), hence once can define an event $e_i^u = (q_i^u, v_i^u, t_i^u)$. Here we note that the Q^u is formed solely for notational convenience, and our method (1) does not require each history item to be associated with a particular query, and (2) does not consider the previous query signals in its next item predictions.

4.4.3.1 Embedding Layer:

Next, we create sets of *learnable embeddings* for items, positions, and search queries, which are then processed by a series of self-attention blocks.

Item Embeddings. An item embedding matrix is denoted as $\mathbf{M} \in \mathbb{R}^{|\mathcal{I}| \times d}$, where d is the latent dimension of item embeddings. The row vector $\mathbf{M_v} \in \mathbb{R}^d$ represents the embedding of an item $v \in I$. A constant zero embedding vector is used for the padding items.

Position Embeddings. A learnable position embedding matrix is denoted as $\mathbf{P} \in \mathbb{R}^{N \times d}$. The row vector $\mathbf{P_k} \in \mathbb{R}^d$ represents the embedding of a position $k \in [1, \cdots, N]$.

Query Embeddings. Since queries are unknown in advance, we need to compute query embeddings on the fly during inference so that we can represent arbitrary and potentially unseen queries. Therefore, one of the standard ways to form a query embedding is to compute the average of its word embeddings [129]: $\mathbf{q} = \frac{\sum_{w \in q} \mathbf{w}}{|q|}$, where $\mathbf{q} \in \mathbb{R}^{d_q}$ and $\mathbf{w} \in \mathbb{R}^{d_q}$ respectively denote the query and word embeddings, and $|q|$ is the length of the query q.

4.4.3.2 Input Layer:

Given an input sequence of interacted items $S^u = (v_1^u, \cdots, v_N^u)$ and queries $Q^u = (q_1^u, \cdots, q_N^u)$, we first right shift the item sequence by one index $\hat{S}^u = (<pad>, v_1^u, \cdots, v_{N-1}^u)$ and then map both sequences to their embedding forms. For $\hat{S}^u$, we combine the embeddings of items and their absolute positions to form input embeddings ($\widehat{\mathbf{E}}$):

$$\widehat{\mathbf{E}} = \begin{bmatrix} \mathbf{0} + \mathbf{P_1} \\ \mathbf{M_{v_1^u}} + \mathbf{P_2} \\ \cdots \\ \mathbf{M_{v_{N-1}^u}} + \mathbf{P_N} \end{bmatrix}, \quad \mathbf{E^q} = \begin{bmatrix} \mathbf{q_1^u} \\ \mathbf{q_2^u} \\ \cdots \\ \mathbf{q_N^u} \end{bmatrix} \tag{4.3}$$

where $\widehat{\mathbf{E}} \in \mathbb{R}^{N \times d}$, and $\mathbf{0}$ is the padding vector. $\mathbf{E^q} \in \mathbb{R}^{N \times d}$ represents the query embedding matrix.

4.4.3.3 History Encoding Layer:

Next, to capture item-item relations, a stack of L self-attention blocks are employed to transform the input embeddings ($\widehat{\mathbf{E}}$) to another latent representation $\mathbf{E^h}$:

$$\begin{aligned} &\mathbf{E^{(0)}} = \widehat{\mathbf{E}} \\ &\mathbf{E^{(l+1)}} = \mathrm{SAB}(\mathbf{E^{(l)}}), \ \ \forall\, l \in [0, \cdots, L-1] \\ &\mathbf{E^h} = \mathbf{E^{(L)}} + \widehat{\mathbf{E}} \end{aligned} \tag{4.4}$$

where $\mathbf{E^h} \in \mathbb{R}^{N \times d}$ is the output of the Lth self-attention block with skip connection to the input embeddings. Being referred as the *item history encoding*, it is essentially a non-linear transformation of input embeddings, where the kth representation ($\mathbf{E^h_k} \in \mathbb{R}^d$) can be seen as a compact summary of the first k interactions, and be used to predict the

$(k+1)$th interacted item [105]. However, such representation alone is not sufficient to fully capture the user intent, since it is still unaware of the search query.

4.4.3.4 Query-Aware History Encoding Layer:

Our next component summarizes the query-relevant parts of user history by capturing query-item relations rooted in data. It consists of an additional attention module, in which the attention weights over the outputs of the history encoding are computed with respect to the search queries. Specifically, we employ a multi-head attention layer [102], which learns attention distributions in h different d/h-dimensional representation subspaces, allowing the model to jointly attend to information from different aspects of the user history.

$$\begin{aligned} &\mathbf{E^{qh}} = Concat(head_1, ..., head_h) \\ &\text{where } head_i = \text{Attention}(\mathbf{E^q} W_i^Q, \mathbf{E^h} W_i^K, \mathbf{E^h} W_i^V) \end{aligned} \tag{4.5}$$

where $W_i^Q \in \mathbb{R}^{d_q \times (d/h)}$, $W_i^K \in \mathbb{R}^{d \times (d/h)}$, $W_i^V \in \mathbb{R}^{d \times (d/h)}$ are projection matrices for each head, and h is the number of attention heads. $\mathbf{E^{qh}} \in \mathbb{R}^{N \times d}$ is called *query-aware history encoding*, summarizing the parts of the interaction history that are most relevant to the search query. Note that we further extend this layer in Section 4.4.4 to incorporate temporal information using a novel multi-resolution attention module.

4.4.3.5 Prediction Layer:

Leveraging all the components introduced so far, we now can predict the next item based on the previous $k-1$ items and the kth search query. In more detail, we combine the representations of the query, the query-aware history encoding and the item history encoding to form a final latent representation:

$$\mathbf{H} = ReLU(Concat(\mathbf{E^q}, \mathbf{E^{qh}}, \mathbf{E^h}))W^H \tag{4.6}$$

where the weight matrix $W^H \in \mathbb{R}^{(d_q+2d)\times d}$ projects the combined representation back into d dimensions. Finally, we measure the relevance score ($r_{k,v_i} \in \mathbb{R}$) of the kth interacted item being $v_i \in \mathcal{I}$ by:

$$r_{k,v_i} = \mathbf{H_k}\mathbf{M}^\top_{\mathbf{v_i}} \tag{4.7}$$

where $\mathbf{H_k} \in \mathbb{R}^d$ is the kth row vector of $\mathbf{H}$, and $\mathbf{M_{v_i}} \in \mathbb{R}^d$ is the embedding of item v_i. Intuitively, items with higher relevance scores are more likely to be interacted, thus we can generate recommendations by ranking the items based on their relevance scores.

4.4.3.6 Optimization:

Recall that we convert the input sequence S^u into a fixed N-length sequence, shifted to the right by one index; $\hat{S}^u = (\textless pad \textgreater, v^u_1, \cdots, v^u_{N-1})$ with the expected output (prediction) sequence being $O^u = S^u = (v^u_1, \cdots, v^u_N)$. In order to learn accurate relevance scores of expected outputs, we use the cross-entropy loss with 100 negative samples at each step:

$$\mathcal{L} = -\sum_{\{\mathcal{E}^u | u\in\mathcal{U}\}} \sum_{k=2}^{N} \left[log(\sigma(r_{k,O^u_k})) + \sum_{v_j \notin O^u} log(1-\sigma(r_{k,v_j})) \right] \tag{4.8}$$

where σ is the *sigmoid* function and $\mathcal{E}^u = (\hat{S}^u, Q^u, T^u, O^u)$ includes the model inputs and expected output for user u. Note that we ignore the first index due to padding. More details on training and implementation are provided in Section 4.5.3.

4.4.4 Multi-Resolution Attention

Modeling input sequences as a combination of item ids and their absolute positions assumes a homogenous temporal resolution across the entire sequence, i.e., time intervals between all adjacent items are the same. However, this is rarely the case in real-world applications [109] as we also demonstrated in Figure 4.2. Motivated by these observations, we now propose a novel approach to incorporate the rich temporal resolution of user history in the setting of personalized item search. To emphasize, we are interested in temporal dependencies between the search query and the past interacted items, unlike the query-less setting where the temporal dependencies are studied solely within the item domain [125, 107, 108, 109, 111].

We introduce a new attention layer—*MultiResAttn*—that is designed to capture asymmetric query-item relations across multiple time resolutions. The main intuition behind our approach is to explicitly guide multiple attention heads to focus on parts of the item sequence that belong to distinct temporal ranges (*i.e.*, *resolutions*).

We first define an attention function $\tilde{A}$ (an adaption of Eq. 4.1) as:

$$\tilde{A}(Q, K, V, C) = \text{Softmax}(\frac{QK^\top + C}{\sqrt{d_{attn}}})V \tag{4.9}$$

where $C \in \mathbb{R}^{N \times N}$ is an additive input to the softmax function, allowing flexibility for controlling (or scaling) the attention weights between queries and items. Note that all upper triangular elements of C are set to $-\infty$ by default to avoid future information leakage. We now explain how we leverage this adaption in our query-aware history encoding layer by modifying Equation 4.5 to take the form:

$$\begin{aligned}
&\mathbf{E^{qh}} = MultiResAttn(\mathbf{E^q}, \mathbf{E^h}, \mathbf{E^h}, \mathbf{T}) \\
&\text{where } MultiResAttn(Q, K, V, T) = Concat(head_1, ..., head_h) \\
&\text{and } head_i = \tilde{A}(QW_i^Q, KW_i^K, VW_i^V, \Phi_i(\mathbf{T}))
\end{aligned} \tag{4.10}$$

with $\Phi_i : \mathbb{R}^{N\times N} \mapsto \mathbb{R}^{N\times N}$ and $\mathbf{T} \in \mathbb{R}^{N\times N}$. $\mathbf{T}$ is a lower triangular matrix including relative time intervals between the search queries and the items; $\mathbf{T}_{k,j} = t_k^q - t_j^v$ (when $k \geq j$), with t_k^q and t_j^v being the timestamps of kth query and jth item, respectively. Recall that $t_k^q = t_{k+1}^v, \forall k \in [1, ..., N-1]$, due to shifted item sequence (Eq. 4.3).

With the help of $\Phi_i(\cdot)$ function, we can enforce certain constraints on $head_i$'s attention distribution, based on $\mathbf{T}$. To this end, we consider two different variants: *(i)* **non-overlapping** and *(ii)* **overlapping** multi-resolution attention. These variants are also illustrated in Figure 4.3 (right). As the names suggest, for the former variant, the time ranges that attention heads cover do not coincide, i.e., $head_1$ covers $[\Delta_0, \Delta_1)$, $head_2$ covers $[\Delta_1, \Delta_2)$ and so on. For the latter variant, each head instead covers an extended range, i.e., $head_1$ covers $[\Delta_0, \Delta_1)$, $head_2$ covers $[\Delta_0, \Delta_2)$ and $head_h$ covers $[\Delta_0, \Delta_h)$. The following $\Phi_i(\mathbf{T})$ function achieves this by masking the items that are out of the desired temporal ranges for $head_i$:

$$\text{overlapping}: \quad \Phi_i(\mathbf{T})_{k,j} = \begin{cases} 0 & \Delta_0 \leq \mathbf{T}_{k,j} < \Delta_i \\ -\infty & otherwise \end{cases} \tag{4.11}$$

$$\text{non-overlapping}: \quad \Phi_i(\mathbf{T})_{k,j} = \begin{cases} 0 & \Delta_{i-1} \leq \mathbf{T}_{k,j} < \Delta_i \\ -\infty & otherwise \end{cases} \tag{4.12}$$

$$\text{s.t. } \Delta_{i-1} < \Delta_i, \ \forall i \in [1, ..., h] \text{ and } \Delta_0 = 0$$

where $\Phi_i(\mathbf{T})_{k,j}$ represents the (k, j)th entry of $\Phi_i(\mathbf{T})$. Note that the time boundaries of attention heads (Δ_i) can be seen as temporal cut-off points in time, which in turn decides on how much representational power is allocated to the respective temporal ranges. A natural choice is to favor most recent interactions with shorter ranges from the search query since they tend to carry a higher influence on users' next interactions [109]. This can be achieved by computing Δs using some form of an exponential function, such as $\Delta_i = ab^i$, where the hyper-parameters $a, b \in \mathbb{R}^+$ are of the same time units as $\mathbf{T}$ (*e.g.* hours, days). While such formulation complies with the exponentially decaying influence phenomenon commonly observed in the literature [130, 111], by varying $\{a, b\}$, one can adapt Δs to different domains with varying temporal resolutions.

Finding good Δs by hyper-parameter tuning can be challenging and may require excessive computational effort. Next, we take our idea a step further and propose a more flexible and adaptive approach. Our goal is to *learn* Δs jointly with the rest of the model. However, the hard-thresholding mechanism (Eq. 4.11 & 4.12) is not differentiable and prevents the model from learning Δs through back-propagation. To sidestep this issue, we propose the following softer-thresholding reparameterization, which remains differentiable with respect to Δs:

$$\Phi_i(\mathbf{T})_{k,j} = \begin{cases} log(\sigma(\frac{\Delta_i - \mathbf{T}_{k,j}}{\tau})) & \text{overlapping} \\ log(\sigma(\frac{\Delta_i - \mathbf{T}_{k,j}}{\tau})) + log(\sigma(\frac{\mathbf{T}_{k,j} - \Delta_{i-1}}{\tau})) & \text{non-overlapping} \end{cases} \tag{4.13}$$

where log and σ denote the natural logarithm and the sigmoid function, while $\tau \in \mathbb{R}^+$ is the temperature scaling parameter.

Taking a closer look into the overlapping variant, the respective item is *masked* when

$\Delta_i - \mathbf{T}_{k,j} << -\tau$ (that is, $\sigma(\frac{\Delta_i - \mathbf{T}_{k,j}}{\tau}) \approx 0$ and $log(\sigma(\frac{\Delta_i - \mathbf{T}_{k,j}}{\tau})) \approx -\inf$). Conversely, it is kept when $\Delta_i - \mathbf{T}_{k,j} >> \tau$ (that is, $\sigma(\frac{\Delta_i - \mathbf{T}_{k,j}}{\tau}) \approx 1$ and $log(\sigma(\frac{\Delta_i - \mathbf{T}_{k,j}}{\tau})) \approx 0$). In other words, the items that are far from Δ_i are either kept or masked based on whether they fall inside or outside of the corresponding boundary. We note that $\partial head_i / \partial \Delta_i \approx 0$ for such items, hence they do not contribute to the learning of Δ_i. On the other hand, the "near boundary" items (*i.e.* $|\Delta_i - \mathbf{T}_{kj,}| \sim \tau$) may or may not be masked depending on their contribution to the final loss, which in turn generates either a pull or a push force on Δ_i. Furthermore, it is straightforward to apply the same logic to the non-overlapping variant, where the second term further masks the items that are already covered by the previous head with boundary Δ_{i-1}. Lastly, some attention heads may have no coverage for certain users who have no interactions within (or near) the respective temporal regions (see Figure 4.2). In such cases, we set $head_i$ to zero vector to indicate the lack of interactions for that particular resolution.

In practice, we initialize Δs using the aforementioned exponential function for faster adaptation and further update them during training. That said, the proposed module is generic and one can choose any increasing function for initializing Δs. More details on training and hyper-parameters are given in Section 4.5.3.

4.5 Experiments

This section introduces our experimental setup, and presents an empirical analysis of our proposed approach. The experiments aim at quantitatively evaluating the contributions of each introduced model component (illustrated in Figure 4.3), as well as comparing our proposed variants with alternative techniques in the literature.

4.5.1 Datasets and Evaluation

Datasets: We evaluate the performance of our method on an open-source benchmark dataset from Amazon [131]. The 5-core version of the dataset is used, where all users and items with less than 5 reviews are removed. Following [94, 105], we treat the presence of a review as an interaction and use the respective timestamps to determine the temporal order of interactions. All other contextual information of items is disregarded to make the data consistent with our setting. We follow the common practice (outlined in [132, 94]) to extract realistic queries for each user-item interaction based on the respective items' hierarchical category information. Although these queries are shown to be similar to real user query formulations in e-commerce platforms [133], we observe that they lead to memorization issues in our setting because each item is always associated with the same query across all users. To alleviate this issue and make the problem more challenging, we randomly drop 50% of the words from the associated query for each user-item interaction recorded in data, leading to more diverse query formulations of the same item across different user sequences.

The following diverse range of categories are employed in our experiments: *Home and Kitchen, Kindle Store, Movies and TV, Pet Supplies, Grocery and Food.* Due to computational constraints, we further remove items with less than 15 interactions for *Home and Kitchen* and less than 10 interactions for *Grocery and Food* category. Dataset statistics are given in Table 4.2.

	Home & Kitchen	*Kindle Store*	*Movies & TV*	*Pet Supplies*	*Grocery & Food*
Number of users	229,210	161,790	250,893	243,690	147,474
Number of items	97,100	153,242	65,860	71,457	44,672
Number of query words	2,946	151	650	1,644	951
Avg. interactions per user	12.94	14.48	10.33	8.62	7.67
Avg. words per query	6.65	4.82	3.39	5.90	4.21

Table 4.2: Statistics of dataset categories.

We follow the same prepossessing steps mentioned in [105]. For users who interacted with at least three items, we use their second last interaction for validation and their last interaction for testing, while the remaining interactions are used for training.

Evaluation Metrics: We evaluate ranking performance by computing Hit@K and NDCG@K with $K \in \{3, 10\}$. Hit@K is a recall-focused metric measuring the percentage of times that the ground-truth next item is among the top K items, while NDCG@K is a position-aware metric which assigns larger weights on higher positions. Following [128, 134], for each user, we sample 100 negative items based on their popularity—excluding the previously interacted items—and rank them together with the ground-truth item.

4.5.2 Baselines

We experiment with a variety of baseline methods ranging from (i) rather simple non-personalized methods to (ii) more sophisticated deep learning based methods for personalized item search, and to (iii) state-of-the-art temporal models adapted from the sequential recommendation literature. These methods are listed below:

- **POP$_i$:** A simple statistical model that ranks items according to their popularity in the training split across all users.

- **Query only (Q):** A query-only approach that ranks items solely based on their respective query embeddings ($\mathbf{E^q}$). We refer to this *non-personalized* approach as Q for simplicity.

- **SasRec (H) [105]:** A position-based self-attention model [105] that ranks items solely based on the item history encodings ($\mathbf{E^h}$). Since it only leverages the user history, it is referred as H for simplicity.

- **SasRec+Q (HQ):** A query-aware approach that ranks items based on the combined signals of **h**istory encodings and **q**uery embeddings, simply referred as HQ (Equation 4.6 without $\mathbf{E^{qh}}$).

The next set of baseline approaches target the modeling of query-aware history encoding ($\mathbf{E}^{qh}$), each extending the HQ variant mentioned above. To that end, we employ two strong approaches [102, 96] for personalized item search and adapt two recently proposed temporal models [109, 111] for sequential recommendation:

- **HQ w/*MultiHeadAttn* [102]:** A benchmark approach that employs standard multi-head attention layer [102].
- **HQ w/*ZeroAttn* [96]:** An approach that employs zero attention mechanism [96], which introduces a zero vector while computing the attention weights between the user query and the user history. This provides the model with the flexibility of paying no attention to user history, allowing for more adaptive personalization.
- **HQ w/*TiSasRec* [109]:** An adaptation of recently proposed TiSasRec [109] model for sequential recommendation. Specifically, following [109], we combine *learnable* relative time interval embeddings with history encodings prior to computing attention weights between the user query and the user history.
- **HQ w/*Dejavu* [111]:** An adaptation of Dejavu [111] model, which is the state-of-the-art for temporal sequential recommendation. Following [111], we employ a mix of time kernels to calibrate the influence (attention weights) of historical actions with respect to the search query, based on the temporal gaps between the two.

As for the proposed model, we experiment with the following two variants introduced in Section 4.4.4:

- **HQ w/*MultiResAttn−O*:** A multi-resolution attention variant where attention heads cover ***overlapping*** temporal ranges (Eq. 4.11).

- **HQ w/*MultiResAttn*:** A multi-resolution attention variant where attention heads cover ***non-overlapping*** temporal ranges (Eq. 4.12).

4.5.3 Model Configurations

We implement the proposed methods and all the baseline approaches using Tensorflow. All the code and data used in our experiments will be made public upon acceptance.

The following settings are applied to each method for fair comparisons. The parameters are learned using mini-batch SGD with Adam optimizer. The {batch size, learning rate, sequence length (N), latent dimensions (d, d_q)} are set to {128, 1e-3, 50, 60}, respectively. The vocab size of search queries is determined based on statistics shown in Table 4.2. The unit of time is set to *days* for all applicable methods. For the proposed variants, we set the Δ initialization parameters $\{a, b\}$ to {1,5} and the temperature scaling parameter τ to 5, which are observed to work well across all datasets. Furthermore, we apply a grid search over the following hyper-parameters on all datasets and applicable methods: number of self-attention blocks (L) in {1,2}, number of attention heads (h) in {1,2,3,4,5}, the vocab size of time embeddings (for *TiSasRecAttn*) in {256,512}, and the number of exponential decay time kernels (for *Dejavu*) in {3,5,10}. The remaining hyper-parameters for baseline approaches are set based on the suggestions made by the authors in their respective papers. Lastly, the best models are selected by early stopping based on the NDCG@10 score on the validation set, with a patience of 20 epochs. All results are reported on the test set.

4.5.4 Experimental Results

Table 4.3 shows the overall performance of baselines and our proposed method variants on all five dataset categories. In this section, unless otherwise stated, the relative

		Baselines								Ours	
Datasets	Metrics	POP$_i$	SasRec (H)	Query only (Q)	HQ	HQ w/ MulHead Attn	HQ w/ Zero Attn	HQ w/ TiSas Rec	HQ w/ Dejavu	HQ w/ MulRes Attn	HQ w/ MulRes Attn−O
Home and Kitchen	Hit@3	0.018	0.117	0.553	0.572	0.578	0.583	0.582	0.594	0.584	**0.601**
	Hit@10	0.067	0.227	0.688	0.715	0.713	0.725	0.720	**0.733**	0.717	0.730
	NDCG@3	0.013	0.096	0.471	0.487	0.497	0.505	0.499	0.508	0.506	**0.519**
	NDCG@10	0.031	0.134	0.521	0.540	0.547	0.557	0.550	0.559	0.555	**0.564**
Kindle Store	Hit@3	0.037	0.484	0.225	0.513	0.542	0.574	0.579	0.572	0.585	**0.596**
	Hit@10	0.107	0.668	0.405	0.728	0.742	0.766	0.764	0.773	0.777	**0.785**
	NDCG@3	0.026	0.408	0.175	0.422	0.447	0.482	0.481	0.474	0.490	**0.505**
	NDCG@10	0.051	0.475	0.239	0.501	0.520	0.552	0.550	0.548	0.561	**0.573**
Movies and TV	Hit@3	0.035	0.290	0.522	0.583	0.625	0.643	0.639	0.636	**0.663**	0.655
	Hit@10	0.118	0.447	0.778	0.815	0.832	0.842	0.847	0.846	**0.852**	0.848
	NDCG@3	0.025	0.244	0.420	0.478	0.521	0.538	0.536	0.533	**0.556**	0.553
	NDCG@10	0.053	0.301	0.514	0.564	0.597	0.611	0.614	0.610	**0.627**	0.623
Pet Supplies	Hit@3	0.022	0.236	0.528	0.566	0.575	0.579	0.591	0.581	0.592	**0.602**
	Hit@10	0.074	0.392	0.714	0.753	0.759	0.776	0.777	0.766	0.771	**0.783**
	NDCG@3	0.015	0.201	0.436	0.471	0.480	0.483	0.496	0.484	0.497	**0.506**
	NDCG@10	0.034	0.256	0.505	0.539	0.548	0.556	0.564	0.553	0.563	**0.573**
Grocery and Food	Hit@3	0.019	0.240	0.675	0.695	0.711	0.720	0.713	0.719	0.728	**0.734**
	Hit@10	0.073	0.354	0.842	0.859	0.865	0.875	0.874	0.872	0.880	**0.883**
	NDCG@3	0.014	0.205	0.570	0.591	0.601	0.619	0.610	0.616	0.625	**0.631**
	NDCG@10	0.033	0.245	0.631	0.652	0.657	0.676	0.671	0.673	0.681	**0.686**

Table 4.3: The ranking performance of baseline and proposed approaches on all five categories. The best performance is highlighted in boldface, while the second best performance is underlined. Results show that our proposed variants consistently outperform the baselines.

performance measures between methods are computed with respect to the NDCG@3 metric.

Ablation results. The first set of four baselines—H, Q, HQ, HQ w/*MulHeadAttn*—helps to assess the incremental contributions of each model component presented in Section 4.4.3, while providing insights into the characteristics of each dataset category. We observe that the query signal alone is more valuable to our task than the user history signal for four of the categories, except Kindle Store. Combining the user history and query signals (see HQ baseline) leads to major improvements compared to the strongest signal of the two across all categories. The largest gain is 13.8% for Movies and TV, while the average gain is 6.4%. HQ w/*MulHeadAttn* baseline leads to further improvements with up to 8.9% relative gain compared to the HQ variant, while the average gain across all categories is 4.1%. This rather sophisticated approach serves as a strong baseline, granted it does not take the temporal aspect of user interactions into account. These results demonstrate the importance of query-aware history summarization for personalized item search and motivate us to investigate further gains when the temporal aspect is considered. For the remaining, we drop the term 'HQ w/' in our referrals to the corresponding methods for simplicity.

Temporal component. Our results reveal a clear trend of approaches with temporal flavor outperforming others that purely rely on sequential patterns. Moreover, our proposed approach consistently achieves best performance across all categories and evaluation metrics. To put this in perspective, among the time-aware methods, our proposed approach outperforms *TiSaSRecAttn* by up to 4.9% and *Dejavu* by up to 6.5%. When compared to *MultiHeadAttn*, we achieve up to 12.9% improvement on ranking performance (*MultiResAttn-O* on Kindle Store). When our best performing variant for each dataset is considered, they collectively provide 6.9% improvement on average compared to *MultiHeadAttn*, which is more than two times the average improvements achieved by *TiSasRecAttn* (3.1%) and *Dejavu* (2.7%) baselines across all datasets. Despite ignoring the temporal signal, *ZeroAttn* overall shows comparable performance to *TiSasRecAttn* and *Dejavu*. Furthermore, both proposed variants also outperform *ZeroAttn* in every comparison.

Between the two proposed variants, the overlapping variant (*MultiResAttn–O*) performs the best for four categories. Movies and TV is the only category where the non-overlapping variant (*MultiResAttn*) achieves a slightly higher ranking than *MultiResAttn–O*. To further investigate the potential motives behind our findings, we take a closer look into

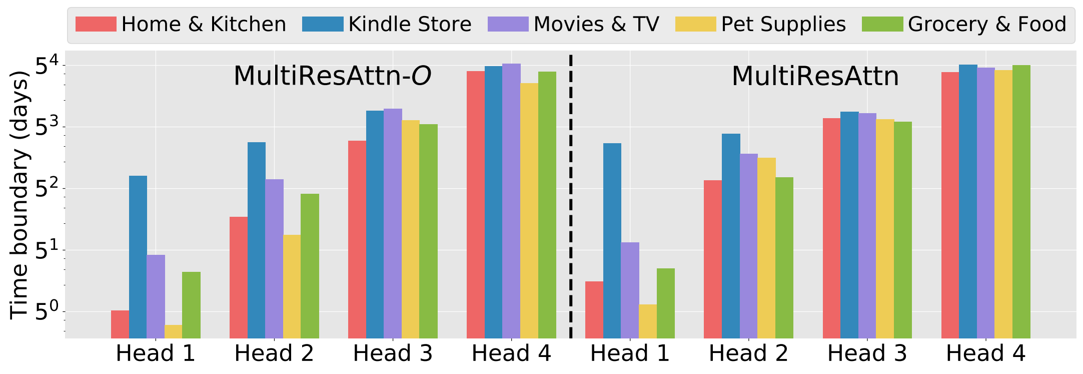


Figure 4.4: Learned time boundaries (Δs) with two proposed variants: MultiResAttn-O (left) and MultiResAttn (right), both with $h = 4$. Time boundaries are plotted on $\log_5$ scale.

the temporal resolutions captured by our proposed variants. Figure 4.4 plots the time boundaries learned by both variants (with h=4) across all categories. We observe higher variations in the learned boundaries for attention heads covering the most recent history (*e.g.*, Heads 1 and 2). In particular, the first time boundary ranges from less than a day (for Pet Supplies) to over a month (for Kindle Store). Moreover, the non-overlapping variant (right) tends to learn slightly longer temporal spans compared to the overlapping variant (left). We conclude that different categories have varying temporal dynamics and densities, and our approach can adaptively recognize such temporal differences found in data.

Sensitivity analysis. Figure 4.5 shows the performance of proposed variants based on the number of attention heads (h). We also include the *MultiHeadAttn* baseline in our analysis for better comparison. For the Movies and TV, the highest score is obtained by the *MultiResAttn* variant with h=3, suggesting that the temporal dependencies are better captured across non-overlapping time spans. On the other hand, the Pet Supplies category favors the $MultiResAttn{-}O$ variant with larger h, implying that the temporal dependencies reach gradually longer time spans that overlap, presumably due to the recurring user needs for this particular category.

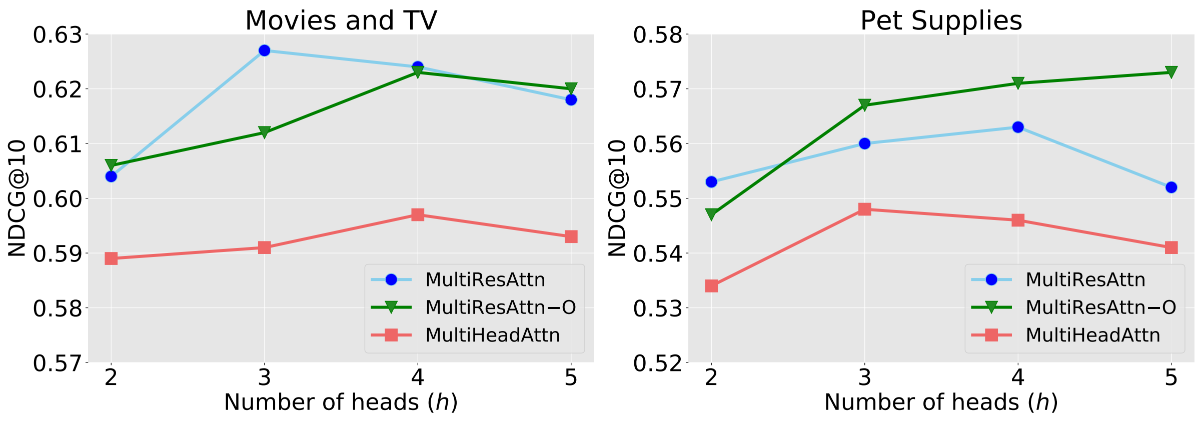


Figure 4.5: Effect of attention head count on ranking performance.

4.6 Conclusion

We propose a Multi-Resolution Attention model for personalized item search. The key component of our architecture is the query-aware history encoding layer, which enables our method to exploit higher-order temporal dependencies between users' search queries and item history. This is achieved by a novel attention module consisting of multiple attention heads, each assigned to recognize and capture users' interests within designated temporal resolutions. The proposed method comes in two variants (*overlapping* and *non-overlapping*) to accommodate different temporal densities of real-world data. Both proposed variants are thoroughly examined by experiments using a large real-world dataset with five different item category domains. Our findings not only demonstrate the efficacy of Multi-Resolution Attention but also provide insights into the varying temporal dynamics captured across different domains. Future work includes applying Multi-Resolution Attention on other forms of temporal data from various problem domains such as query-less temporal recommendation and spatio-temporal learning on graphs.

Chapter 5

FlowGEN: A Generative Model for Flow Graphs

5.1 Introduction

Generative models for graph data have a long-standing history in network science [135, 136]. Major motivations for such models include unavailability of large datasets, due to privacy concerns, cost of data collection, and business interests. Moreover, these models are useful for data-driven discovery, anomaly detection, and large-scale simulations in the natural sciences. While there is an extensive literature on mathematical models for graph data, more recent approaches based on neural generative models have attracted great interest [137, 138, 139, 140, 141, 142, 143, 144, 145].

The main advantage of the recent generative approaches is the potential to learn how to generate a broad class of graphs from a limited number of samples. However, they are primarily focused on reproducing undirected graph topologies observed in some simple graph structures (*e.g.*, grid, community). In the many real-world applications, the ability to mimic (binary) connectivity patterns among nodes alone is oftentimes not sufficient to capture their key characteristics. To that end, there have been recent studies in generative models for other families of graphs that go beyond generic structures, such as molecular graphs [146, 147], directed acyclic graphs (DAGs) [148], city road layouts [149], and program (source code) graphs [150], to name a few.

This paper focuses on the generative modeling of a complex family of graphs called *flow graphs* (FGs). Besides nodes and edges, FGs capture edge flows of a quantity of interest (*e.g.*, water, power, people) being transported through the graph. As these flows often possess higher-order graph-level dynamics driven by sources/destinations [151], hotspots [152], and domain-specific physics [153, 5], generating FGs poses greater challenges than those only related to the graph topology.

As an example, consider a transportation network where nodes correspond to locations, edges are routes between them, and flows indicate the number of people (or vehicles) transported from one location to another. Besides topological properties, such as the existence of central nodes and clusters, this FG also reveals specific flow patterns. For instance, hotspots (*e.g.* recreational areas) might become prominent sources and destinations depending on the time of the day. Moreover, the prevalence of cyclic flows can unveil diurnal variations in travel patterns. The ability to generate realistic mobility flows is key for the understanding of urban dynamics [152] and the spread of epidemics [154]. Similar examples can be constructed for the analysis of load distribution in power networks [155, 156] and flux balance in biological systems [157].

We first investigate how existing graph generative models can be combined with a learnable flow generating function to produce FGs based on observed samples. While this two-step approach is flexible and can be integrated with any existing model, it is unable to learn the joint relationship between graph structure and flows in an end-to-end fashion. To address this limitation, we introduce *FlowGEN*, an implicit generative model for FGs based on the Generative Adversarial Network (GAN) framework. FlowGEN learns to generate both the (directed) graph topology and edge flows. The main ingredient of FlowGEN's architecture is a discriminator with the following components: (1) a permutation invariant flow-pooling layer, (2) bi-directional neural message-passing layers, and (3) an attention-based readout layer. This enables FlowGEN to learn hidden node representations (or states) capturing complex coupling between the graph topology and edge flows.

We assess the performance of flow graph generative models both qualitatively and quantitatively, using novel evaluation metrics designed for flow graphs. Our results show that FlowGEN can effectively reproduce flow properties of a diverse set of real-world and simulated networks, including transportation, power transmission, and water networks, outperforming the alternative approaches.

5.2 Related Work

The generation of realistic graphs that mirror the characteristics of real-world graphs is a long-standing research problem. The earliest graph generative models [135, 136, 158] focus on reproducing certain statistical properties (*e.g.*, degree distribution, graph density) found in real-world graphs using a stochastic generation process. Although simple to use, these models are limited in that the selected properties may not sufficiently represent many other characteristics of real-world graphs.

To address this, inspired by recent advancements in graph neural networks [159, 160], several neural graph generative models have been proposed [137, 138, 139, 140, 141, 142, 143, 144, 161, 145]. Compared to traditional models, neural models are able to learn rather complex structural properties from data (*e.g.*, spectral coefficients and orbit counts), and to reproduce graphs with fundamentally different topologies (*e.g.*, grids and egonets). Amongst the pioneer models, GraphVAE [144] utilizes variational autoencoders

(VAE) [162] to generate a probabilistic fully-connected graph and then applies an expensive graph matching algorithm to calculate the reconstruction loss. As an alternative, NetGAN [137] converts graphs into random walks and learns an implicit probabilistic model for generating walks using adversarial training [163], thus eliminating the need for expensive graph matching. More recently, EDP-GNN [143] leverages score-based generative modeling framework [164] to achieve permutation invariant graph generation.

Another recent paradigm is to model graphs using an auto-regressive process—as a sequence of additions of new nodes and edges, conditioned on the current sub-graph structures [138, 140, 141, 145]. GraphRNN [145] utilizes RNNs for the sequential modeling of nodes, while GRAN [141] considers the sequential modeling of blocks of nodes, employing GNNs with attention [160] for conditioning. BiGG [138] further exploits the sparsity of real-world graphs by combining a recursive edge generation scheme with auto-regressive conditioning, resulting in a technique that sidesteps the explicit generation of each entry in an adjacency matrix. Although these approaches are able to generate larger graphs relative to their counterparts, they rely on pre-defined node orderings of graphs (BFS/DFS) since computing the full likelihood is intractable due to all possible node-orderings. However, this limits their capacity to model long-range (*i.e.* global) dependencies, which is a key element for generating realistic flow graphs as we show in Section 5.4. Moreover, determining a suitable node ordering is subject to the task and data of interest.

Flow graphs are a richer model than simple graphs because they capture the dynamics of a system with node states [165]. The coupling between edge flows and node states is governed by domain-specific physical models, such as the Kirchhoff's laws for power and the LWR model for traffic [166]. Recent work has shown how learning such models from data can improve the prediction of missing edge flows [5]. Here, we focus on learning how to generate flow graphs by implicitly capturing the coupling between topology and flows. Measuring flows (of mobility, water etc.) on graphs requires a lot of effort and resources.

Generative models are especially useful in these settings in order to replicate the behavior of real data for analysis, simulation, and algorithm development.

5.3 Generative Modeling of Flow Graphs

We start by providing some background, and formal definition of our problem.

A **flow matrix** $X \in \mathbb{R}_{+}^{d\times d}$ is a non-negative square matrix where the entry $x_{u,v}$ denotes a *directed flow* from the source node u to destination node v. A **flow graph** $G(V,E,X)$ is a *directed* graph, where V is the set of vertices ($|V| = N$), E is the set of edges, and $X \in \mathbb{R}_{+}^{N\times N}$ is the flow matrix denoting the observed edge flows, where $x_{u,v} = 0, \forall(u,v) \notin E$.

By convention, a negative edge flow indicates that the flow moves in a direction opposite to the direction of the edge. We can always change the sign of the flow to positive as long as we change the edge direction. Therefore, without loss of generality, we assume that all edge flows are non-negative, *i.e.*, $x_{u,v} \geq 0, \forall(u,v) \in E$. Moreover, in cases like human crowd flows, some node pairs are likely to have flows in both directions (say, $\tilde{x}_{u,v} \geq \tilde{x}_{v,u} > 0$). In this work, we are only interested in *net-flows*, that is $x_{u,v} = \max(0, \tilde{x}_{u,v} - \tilde{x}_{v,u})$. We refer to *net-flow* as *flow* throughout this paper.

Problem: Given a set of flow graphs $\{G_1, G_2, \cdots, G_L\}$ sampled from a ground-truth distribution (P_{data}), our aim is to *implicitly* learn to mimic these graphs' complex characteristics such that the graphs (P_θ) generated by our model can simulate both the *graph topology* and the *flow dynamics* found in data. The proposed framework should be generic enough to capture various flow dynamics intrinsic to numerous real-world networks. (Examples of such structures are given in Section 5.4).

Here it is important to emphasize some of the key differences between flow graphs and general weighted graphs. In typical weighted graphs, weights represent the strength of a relationship between two nodes (e.g. homophily in social networks). Conversely, edge flows depend on higher order structures that induce patterns such as source-destinations, cycles and congestion [5].

In the remainder of this Section, we first discuss how existing generative models for undirected (topological) graphs can be adapted to our problem. Next, we introduce an implicit flow graph generative model (named *FlowGEN*) that is designed to capture graph structure as well as complex flow characteristics observed in real data, being the first end-to-end graph generative model to generate FGs from a domain-agnostic standpoint.

5.3.1 A Two-step Approach for Generating Flow Graphs

Whether one generates the entire graph at once [137] or as a sequential process of adding new nodes and edges [145], we can think of most (undirected) graph generative models as a parametric function $\mathcal{H} : \mathcal{Z} \mapsto \{0,1\}^{N\times N}$ (typically a neural network of some kind) that given samples drawn from a prior distribution, learns to generate graphs, ultimately in the form of an adjacency matrix or its counterparts. It is possible to adapt these models to the generation of flow graphs by a two-step model in which the second step learns an additional flow function $\mathcal{F} : \{0,1\}^{N\times N}\mapsto\mathbb{R}^{N\times N}$ that takes an adjacency matrix A as input and generates a non-negative flow matrix whose values correspond to the non-zero elements of A. Within this two-step framework (that can be summarized as $\mathcal{F}\circ\mathcal{H}$), one can choose to use any undirected graph generative model as function $\mathcal{H}$ that is learnt on undirected versions of the observed FGs. We focus on the learning of $\mathcal{F}$, as illustrated in Figure 5.1.

The problem of learning $\mathcal{F}$ is closely related to flow estimation, which can be formulated as a semi-supervised learning problem on graphs [159]. This problem was most recently studied by [153, 5] for a single FG snapshot, where the goal is to predict missing edge

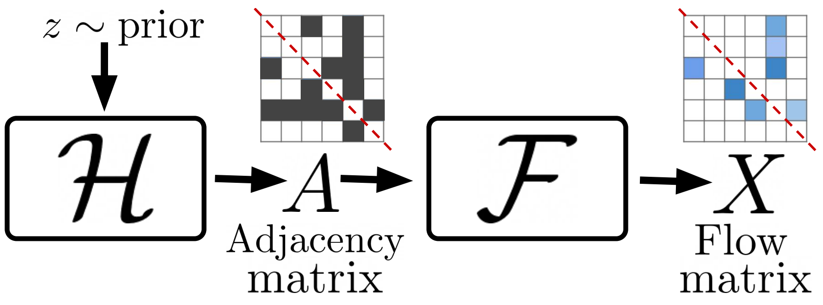


Figure 5.1: An illustration of our two-step approach for generating flow graphs.

flows based on the graph topology and partial flow observations. [153] achieves this by solving a constrained optimization problem involving domain-specific physical constraints (flow conservation law), while [5] further relaxes these constraints and instead enforces them as a regularizer whose weights are learned based on various node/edge features. However, these methods cannot be applied to our setting directly because we do not assume access to other information such as partial flows (or features), and we aim to learn generative models in a domain-agnostic manner based purely on data.

We now discuss various ways of constructing $\mathcal{F}$. Similar to [161], given a graph in the form of an adjacency matrix $A \in \{0,1\}^{|V|\times|V|}$, we first generate node features $\mathcal{M} \in \mathbb{R}^{|V|\times d_k}$ as a standard low-dimensional spectral embedding [167] based on A. Notice that this transformation allows us to handle variable sized graphs. Next, we learn node states based on the node features using a standard two-layer MLP:

$$H = MLP(\mathcal{M}) = ReLU(\mathcal{M}W_0)W_1 \tag{5.1}$$

Alternatively, we can better utilize the graph topology by using a GCN model instead of MLP:

$$H = GCN(\mathcal{M}, A) = \tilde{A}ReLU(\tilde{A}\mathcal{M}W_0)W_1 \tag{5.2}$$

where $H \in \mathbb{R}^{|V|\times d_l}$, $W_0 \in \mathbb{R}^{d_k\times d_l}$, and $W_1 \in \mathbb{R}^{d_l\times d_l}$. $\tilde{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ is symmetric normalized adjacency matrix of A, and D is the diagonal degree matrix with $D_{ii} = \sum_{j=1}^{|V|} A_{ij}$. Lastly, we adapt a final linear transformation to predict edge flows based on the topological encoding of source ($h_s \in \mathbb{R}^{d_l}$) and target ($h_t \in \mathbb{R}^{d_l}$) nodes:

$$\mathbf{g}(h_s, h_t) = (h_s - h_t)W_2 \tag{5.3}$$

where $W_2 \in \mathbb{R}^{d_l \times 1}$, and $h_i = H_{i,:}$ for $i \in V$. The bias terms are omitted for convenience.

Notice that $\mathbf{g}(h_s, h_t)$ intrinsically captures the *edge direction* between nodes s and t, where $\mathbf{g}(h_s, h_t) = -\mathbf{g}(h_t, h_s)$. This ensures consistency with our problem setting, *i.e.*, a negative flow means movement against the orientation of the edge. Therefore, the respective directions of non-negative flows are assigned for each node pair $(s, t) \in E$, such that $s \rightarrow t$ when $\mathbf{g}(h_s, h_t) > \mathbf{g}(h_t, h_s)$.

Finally, we can define the flow function $\mathcal{F}$ as a stack of $\mathbf{g}$ functions applied on each edge of the input graph. All layers of $\mathcal{F}$ can be trained jointly using graphs from P_{data}, with MSE loss computed between predicted ($\mathbf{g}(h_s, h_t)$) and ground-truth ($x_{s,t}$) flows. We refer to our supplementary materials for all training details including the formulation of the loss function, hyper-parameters, and more.

5.3.2 FlowGEN Framework

Though the two-step approach presented above can reuse existing graph generative models, it does not account for the joint relationship between graph structure and flow distribution. To this end, we introduce FlowGEN (illustrated in Figure 5.2), an implicit **flow** graph **gen**erative model that can generate FGs in an end-to-end fashion based on the GAN framework [163]. As in the standard GAN framework, our model consists of two components trained jointly: (1) A *generator* $\mathcal{G}_\theta$ that takes a sample from a prior distribution and learns to generate flow graph samples that approximate those drawn

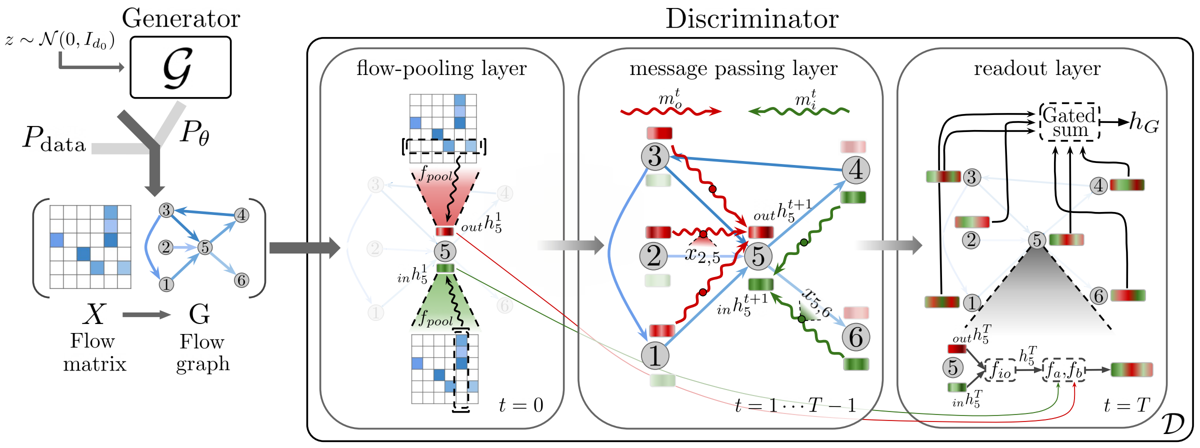


Figure 5.2: A high-level illustration of FlowGEN architecture (*best viewed in color*). The left part pictures the adversarial training process. The right part shows a detailed sketch of our discriminator: (1) The flow pooling layer assigns two initial feature vectors (*in* and *out*) to each node in a permutation-invariant manner. (2) The message-passing layers comprise bi-directional channels that exchange *in* and *out flow* messages (spiral arrows) *independently* in opposing directions. (3) The read-out layer fuses *in* and *out* states of each node, and further aggregates them to construct a *global* flow graph representation. All components are trained jointly. See Section 5.3.2 for details.

from the dataset, and (2) a *discriminator* $\mathcal{D}_\phi$ that learns to distinguish whether an input graph is sampled from the dataset or the generator.

5.3.2.1 Generator

The generator $\mathcal{G}_\theta$ (parameterized by θ) takes as input a random vector $z \in \mathbb{R}^{d_0}$ sampled from $\mathcal{N}(0, I_{d_0})$ and outputs a flow graph. Precisely, a flow matrix $X \in \mathbb{R}_+^{N\times N}$ is computed by passing the sampled vector z through multiple layers of fully connected neural networks (MLP):

$$X = ReLU\left(f_\theta^k \circ \cdots \circ f_\theta^1(z)\right) \tag{5.4}$$

where $\circ$ represents function composition. We use ReLU activations for all layers including the final layer to obtain non-negative flow matrix. Note that the mapping $\theta \mapsto P_\theta$ is continuous (P_θ denotes the model distribution) which differentiates our problem space from the discrete graph generative models [145, 141].

5.3.2.2 Discriminator

The discriminator $\mathcal{D}_\phi$ (parameterized by ϕ) takes a flow graph as input and outputs a scalar value indicating whether an input graph is likely to be sampled from the dataset or the generator. Our discriminator is a customized (flow) graph neural network (GNN) consisting of three main components: (1) permutation invariant flow-pooling layer, (2) bi-directional neural message-passing layers, and (3) an attention-based readout layer, each designed to collectively account for domain-agnostic and complex flow dynamics in directed flow graphs.

More formally, given a flow graph $G(V, E, X)$, we first compute two node-level flow (feature) vectors—*in* and *out*—using a permutation invariant flow-pooling function over the column and row vectors of the flow matrix, respectively.

$$ {}_{in}h^1_u = f_{pool}(X_{:,u}) \ , \ {}_{out}h^1_u = f_{pool}(X_{u,:}) \ \ u \in V \tag{5.5} $$

Here, $f_{pool}(x) = [\text{MEAN}(x); \text{MAX}(x); \text{SUM}(x)]$ is a stack of element-wise pooling aggregators—the symbol ';' denotes concatenation and $x \in \mathbb{R}^d$. $X_{:,u}$ and $X_{u,:} \in \mathbb{R}^{|V|}$ indicate the columns and rows of X, which, respectively, hold the incoming and outgoing flow information of node u from and to its neighbors. Node flow vectors ${}_{in}h^1_v$ and ${}_{out}h^1_v \in \mathbb{R}^{d_1}$ ($d_1 = 3$ in this case) are then used as initial hidden states to *in* and *out* neural message-passing channels, where respective node updates are computed independently as follows:

$$ {}_{in}h^{t+1}_u = f^t_i({}_{in}h^t_u, \sum_{v \in G_{out}(u)} m^t_i({}_{in}h^t_v, x_{u,v})) \tag{5.6} $$

$$ {}_{out}h^{t+1}_u = f^t_o({}_{out}h^t_u, \sum_{v \in G_{in}(u)} m^t_o({}_{out}h^t_v, x_{v,u})) \tag{5.7} $$

where, for each node u at layer $t \in \{1, \cdots, T-1\}$, the *in* and *out* message functions (m_i^t and m_o^t), respectively take as input the concatenation of *out* and *in* neighbors' corresponding to hidden states and edge flow values to compute latent *flow message* vectors. The *in* and *out* node update functions (f_i^t and f_o^t) take as input the concatenation of current *in* and *out*-states of u and aggregated flow messages from its neighbors to update its next hidden states ${}_{in}h_u^{t+1}$ and ${}_{out}h_u^{t+1} \in \mathbb{R}^{d_{t+1}}$ accordingly. $\{f_i^t,\ m_i^t,\ f_o^t,\ m_o^t | t = 1, \cdots, T-1\}$ are all modeled independently as MLPs, with no weight sharing.

After $T-1$ message passing layers, we apply another node update function, which takes as input the *in* and *out* node states, and combines them into final node representations.

$$h_u^T = f_{io}({}_{in}h_u^T\ ,\ {}_{out}h_u^T)\ \ u \in V \tag{5.8}$$

where f_{io} is another small MLP. The vector $h_u^T \in \mathbb{R}^{d_T}$ is a fused latent representation of node u, which captures its *in* and *out* flow dynamics conditioned on its neighbors as well as its relative location in the graph.

We aggregate these node representations and obtain a latent graph representation $h_G \in \mathbb{R}^{d_G}$ using an attention-based readout layer (a.k.a. *gated sum*) following [168]:

$$h_G = \sum_{u \in G} \sigma(f_a(\tilde{h}_u))\ \odot\ f_b(\tilde{h}_u) \tag{5.9}$$

where $\tilde{h}_u = [h_u^T\ ;\ h_u^1] \in \mathbb{R}^{d_T + 2d_1}$ and $h_u^1 = [{}_{in}h_u^1\ ;\ {}_{out}h_u^1] \in \mathbb{R}^{2d_1}$ is the concatenation of initial *in* and *out* flow vectors (Eq. 5.5). This is loosely analogous to skip connections in residual networks [169] which we empirically find to be effective in capturing global flow statistics. The first term in the summation (Eq. 5.9) essentially serves as a soft attention mechanism assigning contextual importance scores to nodes, where σ is the *sigmoid* activation and $\odot$ denotes element-wise multiplication. The context functions f_a

and f_b are again modeled as small MLPs. Note that such gated sum can model injective multiset functions, and is invariant to input (node) order. Lastly, h_G is processed by a final MLP layer (f_G) which outputs a graph-level scalar $\in \mathbb{R}$. Model parameters $\{\theta, \phi\}$ are trained jointly using the Wasserstein GAN objective [170] with gradient penalty [171]. For all training details (hyper-parameters, activation functions etc.), see the supplementary materials.

5.3.3 Discussion

We now present a brief rationale behind the design of some of FlowGEN's components. Since we generate the entire flow graph at once, this limits us to graphs whose sizes do not exceed a certain threshold. However, generating an entire flow graph at once is more appropriate for capturing global statistics of flows (*e.g.* number of nodes with conserved flows) and graph topology (*e.g.* number of k-cycles), as well as correlations between the two (*e.g.* hotspot nodes and their relative locations in the graph). That said, our model is capable of generating graphs with varying number of edges and fewer nodes than the predetermined maximum (by removing disconnected nodes). In addition, since f_{pool} is invariant to permutations of the flow matrix and can handle flow distributions of varying sizes, similar to GNNs, our discriminator also remains permutation invariant and is able to model variable-sized flow graphs. Moreover, our framework is general enough to incorporate other more sophisticated flow-pooling functions and neighbor aggregation functions [160]. One potential shortcoming of FlowGEN lies in its generator, which has a quadratic memory print in the number of nodes, similar to other non-autoregressive models [137, 142, 143]. Here we emphasize that our primary goal is to generate high-quality flow graphs. For large flow graphs, one can resort to the proposed two-step approach by leveraging a more scalable topological graph generative model [138].

5.4 Experiments

In this section we present an experimental analysis of the proposed approaches on both synthetic and real graph datasets with diverse flow characteristics. Due to space constraints, additional results are presented in the supplementary materials.

5.4.1 Experimental Setup

Datasets. Here, we briefly describe the datasets used in our experiments. See supplementary materials for more details.

(1) Power: European power transmission graphs [172] where nodes ($|V|$=25) represent power infrastructure (buses, plants etc.) belonging to European countries, edges ($|E|_{avg}$=45) represent transmission lines between the countries, and edge flows measure the **total active power** being transmitted through these lines. We randomly sample 2000 hourly snapshots from the year 2013.

(2) Water: 1000 directed barbell graphs with 20 nodes and 10-40% of edges inside the communities removed uniformly at random. Flows are synthetically generated based on a simple **water distribution** process with either a *source* or a *sink* node assigned to each community. For each node pair (u, v), the flow is the *net* amount of water that passed through either direction during the simulation.

(3) Taxi Trips: Real flow graphs with nodes and edges representing taxi zones in Manhattan, NYC, and routes between zones, respectively. Flows are net **volumes of passengers** traveled between zones during 584 different snapshots (weekdays in [2017,2019]). We create two versions of this dataset, with data from **8-9am** ($|V|_{max}$=32, $|E|_{avg}\approx$133.1) and **6-7pm** ($|V|_{max}$=32, $|E|_{avg}\approx$156.2).

Evaluation Metrics. Evaluating generative models is known to be challenging [173]. In our case, this evaluation requires a comparison between the *generated* and the *ground* *truth* flow graphs while accounting for both the graph structures and edge flow characteristics. To this end, we identify a comprehensive set of metrics which allow us to quantitatively evaluate the effectiveness of our model on this novel task. These metrics include: **(1-2)** *in/out degree distributions*, **(3)** *edge flow distributions*, **(4)** *node divergence distributions*, and **(5-6)** *number of directed k-cycles* ($k \in \{3, 4\}$). We define the *divergence* on node u as the difference between total in-flow and total out-flow, normalized by their maximum, in order to compare graphs across different datasets. Formally, $X_u^{div} = (X_u^{out} - X_u^{in})/max(X_u^{out}, X_u^{in})$ where $X_u^{out} = \sum_{\{u \to v\}} x_{u,v}$ and $X_u^{in} = \sum_{\{v \to u\}} x_{v,u}$. Note that $X_u^{div} \in [-1, 1]$ for all $u \in V$ and node u is a *source* (*sink*) node of the flow graph if $X_u^{div} \approx 1$ ($X_u^{div} \approx -1$). There can be multiple source and sink nodes in a graph.

For graph-level metrics such as number of directed k-cycles, we compute the average statistics of the entire set. For the remaining node-level and edge-level metrics, following previous work [145, 141], we compute the maximum mean discrepancy (MMD) over the two sets of distributions using the total variation (TV) distance.

Methods. To the best of our knowledge, no baseline is available for the novel task of generating FGs that exhibit the characteristics of a given set of (ground-truth) graphs. Therefore, we experiment with approaches introduced in Section 5.3 including FlowGEN and a variety of two-step alternatives. Following our discussion on two-step approaches (Section 5.3.1), we consider two state-of-the-art deep (undirected) graph generative models —NetGAN [137] and GRAN [141]— as the first step (function $\mathcal{H}$), which are trained on the undirected versions of the datasets. As for the second step (function $\mathcal{F}$), we implement three approaches namely FMLP (based on Equation 5.1), FGCN (based on Equation 5.2), and DFF (short for Divergence-Free Flows proposed by [153]). This results in six variants: *NetGAN-FMLP, NetGAN-FGCN, NetGAN-DFF, GRAN-FMLP, GRAN-FGCN, GRAN-DFF*. More details on these methods are provided in the supplementary materials due to space limitations.

Experiment settings. Following [141], for each dataset, we randomly split the graphs into train (80%) and test (20%) sets. We use 20% of the training graphs as the validation set. For fair comparison, we use the same splits for all the models, and each model—at testing time—generates the same number of graph samples as the test set. For all the methods, we fix the size of input and output graphs as the size of the largest graph in the dataset. All evaluations are performed on the test set.

5.4.2 Experimental Results

Table 5.1 reports the performance of all seven methods on all four datasets. The set of metrics we use evaluates graphs from varying perspectives and the proposed end-to-end model consistently displays top results on all of them with very few exceptions. In more detail, we observe that:
(i) FlowGEN excels at capturing global cyclic trends in all datasets compared to other methods. This demonstrates that FlowGEN is able to successfully replicate various higher-order patterns of directed graphs including acylic water flows and real human-crowd flows at different rush hours of the day.
(ii) FlowGEN fits the underlying edge flow and node divergence distributions in the data considerably better than the competing methods on nearly all the measures. This suggests that while FlowGEN is effective in learning the edge flow distributions directly from data, it also learns the *coupling between edge flows* and *graph topology* which is intrinsically tied to the node-divergence distribution. We attribute this ability to our bi-directional message-passing layers along with contributions of other components as shown by an ablation study in the supplementary materials (Section 5.6.8).
(iii) GRAN-based two-step variants are particularly good at recovering degree statistics and often outperform other methods regarding in/out-degree distributions. This is a

			In-deg. dist.	Out-deg. dist.	$\{x_{u,v}\}$	$\{X_u^{div}\}$	△	□	Avg. rank
Simulated Data	**Power**	test data	–	–	–	–	2.58	2.33	–
		NetGAN-FMLP	$1.69e^{-2}$	$3.73e^{-2}$	$4.71e^{-2}$	$7.26e^{-2}$	2.21	1.45	5.2
		NetGAN-FGCN	$1.88e^{-2}$	$1.21e^{-2}$	$1.89e^{-2}$	$5.95e^{-2}$	2.01	1.77	3.6
		NetGAN-DFF	$5.85e^{-2}$	$2.76e^{-2}$	$4.22e^{-2}$	0.126	3.74	5.01	5.8
		GRAN-FMLP	$3.16e^{-3}$	$2.28e^{-3}$	$3.94e^{-2}$	$5.62e^{-2}$	4.24	3.16	3.6
		GRAN-FGCN	**$1.18e^{-3}$**	**$1.79e^{-3}$**	$7.02e^{-3}$	$4.51e^{-2}$	3.98	3.14	2.3
		GRAN-DFF	$1.56e^{-2}$	$1.73e^{-2}$	$3.61e^{-2}$	0.164	7.23	6.62	5.6
		FlowGEN	$1.35e^{-2}$	$9.36e^{-3}$	**$3.73e^{-3}$**	**$2.98e^{-3}$**	**2.66**	**2.41**	**1.6**
	Water	test data	–	–	–	–	0.0	0.0	–
		NetGAN-FMLP	$4.92e^{-2}$	$4.53e^{-2}$	0.107	0.497	5.58	6.89	5.3
		NetGAN-FGCN	$2.65e^{-2}$	$2.16e^{-2}$	$8.94e^{-2}$	0.419	4.84	6.07	3.2
		NetGAN-DFF	$7.52e^{-2}$	$7.64e^{-2}$	$8.99e^{-2}$	0.264	9.16	12.87	5.3
		GRAN-FMLP	$3.91e^{-2}$	$4.18e^{-2}$	$9.98e^{-2}$	0.464	9.78	14.96	3.5
		GRAN-FGCN	**$1.81e^{-2}$**	$1.78e^{-2}$	$8.51e^{-2}$	0.408	8.23	12.56	3.0
		GRAN-DFF	$3.66e^{-2}$	$4.35e^{-2}$	$8.16e^{-2}$	0.235	13.54	25.32	4.5
		FlowGEN	$1.99e^{-2}$	**$1.76e^{-2}$**	**$2.41e^{-2}$**	**$5.53e^{-2}$**	**3.39**	**5.82**	**1.2**
Taxi Trips	**8-9am**	test data	–	–	–	–	10.34	17.71	–
		NetGAN-FMLP	$3.01e^{-2}$	$4.45e^{-2}$	$9.81e^{-2}$	$6.62e^{-2}$	2.56	5.68	4.5
		NetGAN-FGCN	$3.56e^{-2}$	$2.49e^{-2}$	**$5.32e^{-2}$**	$3.73e^{-2}$	6.74	**10.85**	2.6
		NetGAN-DFF	0.128	$4.76e^{-2}$	$8.51e^{-2}$	0.322	26.41	67.84	5.8
		GRAN-FMLP	$1.96e^{-2}$	$2.33e^{-2}$	0.122	$5.45e^{-2}$	17.53	55.37	3.8
		GRAN-FGCN	$2.28e^{-2}$	**$9.81e^{-3}$**	$6.77e^{-2}$	$2.85e^{-2}$	34.19	110.51	3.3
		GRAN-DFF	$8.97e^{-2}$	$1.11e^{-2}$	0.101	0.315	61.24	263.43	5.6
		FlowGEN	**$1.42e^{-2}$**	$3.47e^{-2}$	$8.32e^{-2}$	**$1.99e^{-2}$**	**11.76**	28.02	**2.1**
	6-7pm	test data	–	–	–	–	34.10	92.56	–
		NetGAN-FMLP	$6.44e^{-2}$	$1.27e^{-2}$	$3.07e^{-2}$	$4.32e^{-2}$	8.59	20.25	5.2
		NetGAN-FGCN	$4.09e^{-2}$	$1.32e^{-2}$	$2.81e^{-2}$	$2.96e^{-2}$	11.45	27.02	4.2
		NetGAN-DFF	$5.03e^{-2}$	$5.67e^{-2}$	$9.34e^{-2}$	0.243	**28.36**	72.30	4.8
		GRAN-FMLP	$3.14e^{-2}$	**$5.98e^{-3}$**	$3.04e^{-2}$	$2.44e^{-2}$	22.17	75.58	2.6
		GRAN-FGCN	$1.39e^{-2}$	$8.76e^{-3}$	**$2.66e^{-2}$**	$2.21e^{-2}$	53.21	205.41	3.0
		GRAN-DFF	**$5.12e^{-3}$**	$4.27e^{-2}$	0.140	0.289	95.18	417.35	5.8
		FlowGEN	$6.77e^{-3}$	$1.23e^{-2}$	$7.44e^{-2}$	**$1.93e^{-2}$**	23.69	**81.48**	**2.3**

Table 5.1: Flow graph generation results. Metrics from left to right: (1-2) in&out degree distribution, (3) edge flow distribution, (4) node divergence distribution, (5-6) average number of directed 3&4-cycles. For MMD scores (1-4), the smaller the better; for average statistics (5-6), the closer to test data, the better. Last column (7) shows average rank of models per each dataset for the reader's convenience. The symbol '–' means not applicable as MMD scores are computed with respect to test data. FlowGEN consistently ranks top across all four datasets by showing superior performance on flow-related metrics (3-6), while being competitive with other methods on topological metrics (1-2).

natural outcome given that GRAN is considered to be the state-of-the-art (undirected) graph generative model. FlowGEN, on the other hand, has competitive—if not better, as for Water—scores than GRAN variants regarding these metrics, while consistently outperforming them on flow-related metrics.

To perform visual inspection, we also display graph examples generated by FlowGEN and the best-performing two-step method, together with training samples for the taxi trips datasets in Figure 5.3. We can observe that, in addition to generating similar graph structures to those in the training set, FlowGEN remarkably exhibits similar numbers of *source (green)* and *sink (red)* nodes (as opposed to divergence-free nodes) together with their relative locations in the graph.

For instance, the morning hour exhibits unique flow patterns where the vast majority of mobility flow is directed towards a few high centrality nodes corresponding to business centers in Manhattan. Conversely, during the evening hour, such nodes become the source of mobility flows being directed towards lower centrality (or periphery) nodes, coupled with more cycles (also see Table 5.1) due to variations in travel patterns.

To further illustrate these findings, Figure 5.4 plots node-level flow divergence distributions of graphs randomly sampled from training data (1-5), FlowGEN (6-10), NetGAN-FGCN (11-15), and GRAN-FGCN (16-20) for both dataset variants, with the top plot corresponding to 8-9am and the bottom plot corresponding to 6-7pm. The results show that FlowGEN consistently matches the ground-truth distribution observed in both settings of the data, while two-step approaches NetGAN-FGCN and GRAN-FGCN struggle to reproduce such complex characteristics. Thus, we emphasize that the ability to capture the joint relationship between underlying graph topology and global flow distribution is one of the key desiderata in generating *realistic* flow graphs observed in diverse settings.

Through both qualitative and quantitative analyses, we conclude that FlowGEN can effectively capture the flow dynamics of graphs with vastly differing characteristics—being

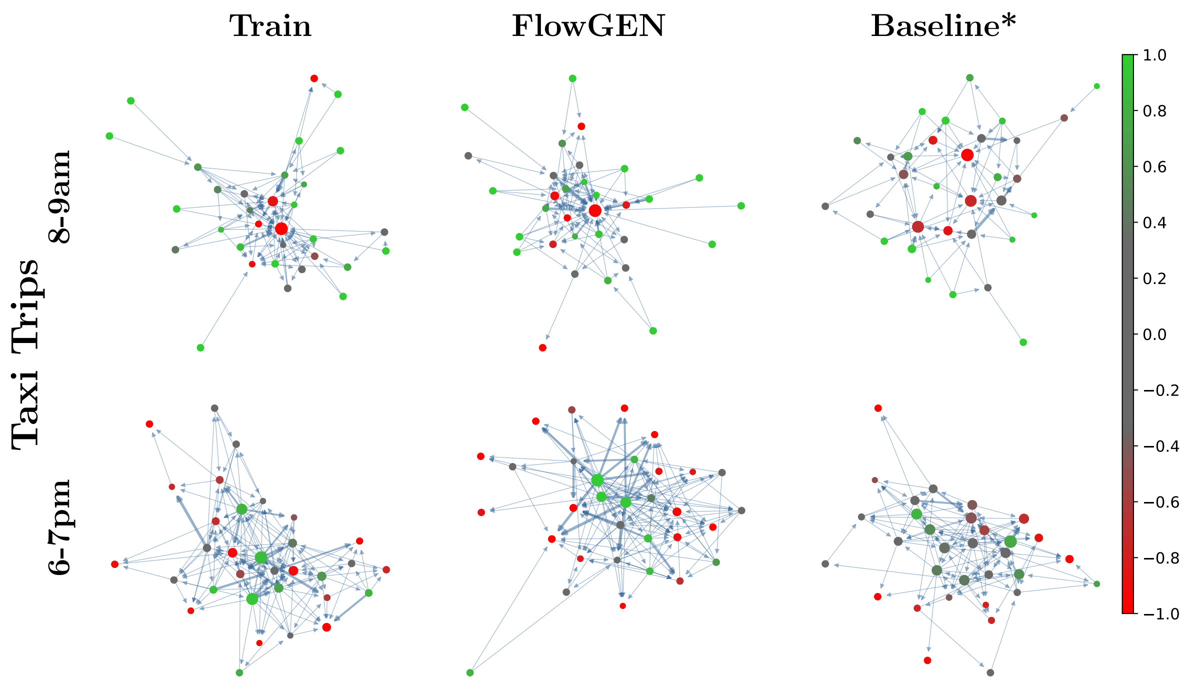


Figure 5.3: Visualization of flow graphs (FGs) sampled from train splits and top-2 models for taxi trips. '*' indicates the best performing baseline variant (*e.g.*, *NetGAN-FGCN* for *8-9am*). Node colors represent *flow divergence* with *green* for *source*, *red* for *sink* and *gray* for *divergence-free* node. Node sizes indicate total in/out flow normalized per graph. See color map on the right for scale. FGs generated by FlowGEN exhibit similar patterns to those observed in data regarding the coupling of graph structure and flow distributions.

able to learn complex distributions like divergence-free flows as well as more natural distributions of flows like human crowds and power transmissions. Lastly, we refer to our supplementary materials for results on other graph statistics (clustering coefficient, degree assortativity etc.), additional comparisons with traditional graph generation models (Section 5.6.4), diversity analysis (Section 5.6.9), as well as a detailed ablation study of our discriminator components, demonstrating their contributions to each of the evaluation metrics (Section 5.6.8).

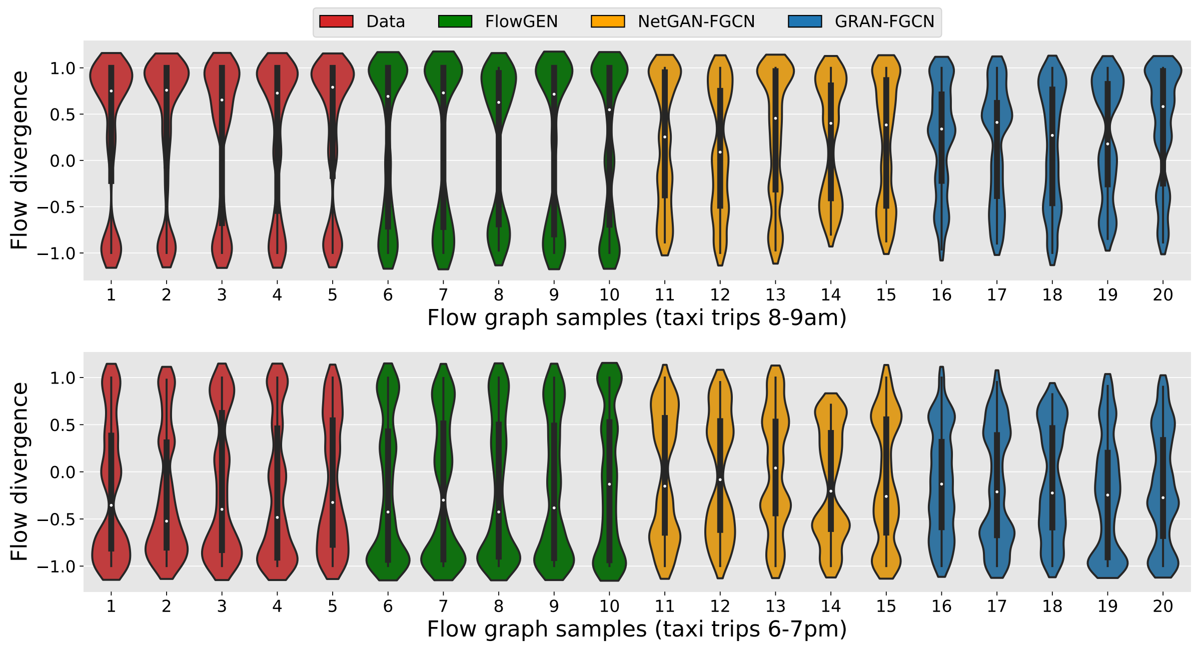


Figure 5.4: Node-level flow divergence distributions of randomly sampled graphs from training data (1-5), FlowGEN (6-10), NetGAN-FGCN (11-15), and GRAN-FGCN (16-20) for taxi trips 8-9am (top) and 6-7pm (bottom) variants. The horizontal axis corresponds to color-coded graph samples (five each), while the vertical axis represents flow divergence distributions. FlowGEN successfully replicates the flow divergence distribution observed in data, outperforming the two-step approaches.

5.5 Conclusion

We have introduced FlowGEN a generative model for flow graphs (FGs). Besides nodes and edges, FGs also capture edge flows that might represent a quantity of interest (e.g. water, power, people) being transported through the graph. FlowGEN generates both the graph topology and edge flows using a graph neural network customized for flow data. In particular, FlowGEN's discriminator has the following components: (1) a permutation invariant flow-pooling layer, (2) bi-directional neural message-passing layers, and (3) an attention-based readout layer. We have evaluated our model against alternative approaches using synthetic and real datasets and based on a diverse set of metrics. Results have shown that FlowGEN is able to capture complex flow statistics, such as flow cycles and flow conservation. Moreover, FGs generated by FlowGEN are also match the number of source and sink nodes (or hotspots) and their relative locations with those found in the input graphs.

This work spurs promising directions for future inquiry. First, can one generate flow graphs with potentially larger sizes than the input ones? A possible approach is to generate flow graphs hierarchically, where the flow dynamics is induced at multiple (spatial) scales. Second, the ability of our discriminator to capture complex properties of flow graphs shows that it can also be applied to other downstream tasks, such as flow graph classification and missing flow prediction. More broadly, our work might also inspire future research on neural generative models for other types of graphs with relevant higher order properties (e.g. structurally balanced opinion graphs), for which there is no evidence that traditional graph generative models can be successfully applied.

5.6 Supplementary Material

Our supplementary material is organized as follows. Section 5.6.1 provides details on the datasets employed in our experiments. Sections 5.6.2 and 5.6.3 respectively provide training details for FlowGEN and two-step approaches. More experimental results including traditional graph generation models and other graph topological statistics are given in Section 5.6.4. Section 5.6.5 discusses the stopping criterion applied during training. Section 5.6.6 presents a comprehensive case study demonstrating FlowGEN's ability to generate power networks with domain-specific properties. Section 5.6.7 compares runtimes of all models. Section 5.6.8 presents an ablation study of our discriminator components. Lastly, Section 5.6.9 provides a diversity analysis of the generated graphs.

5.6.1 Dataset Details

Here we provide more details on the datasets used during our experiments.

Power: The dataset consists of a fixed power network topology and a set of hourly snapshots with varying edge flows. Nodes ($|V|$=25) represent power infrastructure (buses, plants etc.) belonging to European countries, edges ($|E|_{avg}$=45) represent transmission lines between the countries, and edge flows measure the total active power (in MW) being transmitted through these lines. We randomly sample 2000 hourly flow graph snapshots from the year 2013. The dataset is obtained from PyPSA-Eur [172]—an optimization model of the European power transmission system—which generates realistic power flows based on solutions of optimal linear power flow problems with historical production and consumption data. Default values were applied for the PyPSA-Eur settings.

Water: This dataset consists of 1000 *barbell* graphs, each consisting of two *densely connected* communities (belts) of size 9, connected by a path of size 2, totalling 20 nodes. To vary graph structure between samples, instead of having two *complete* graphs as left and right bells as in [174], we remove a ratio $r_i \sim U(0.1, 0.4)$ of the *intra-community edges* from each graph G_i. Removed edges are selected uniformly at random and removals that disconnect the graph are not allowed. Having generated the graph structures we now regard them as ***2D pipe networks*** and simulate ***water flows*** between a source and a sink node, each selected uniformly at random from different communities. For each graph G_i, we inject a $w_i \sim U(1, 20)$ unit of water from its source node and simulate the water flow until all the injected water leaks from its sink node. Then, edge flows ($x_{u,v}$) are assigned as net amounts of water passing through that edge during the simulation. All edges are assumed to have the same capacity $c_i \gg w_i$ and length. Note that other than the source and sink nodes, remaining nodes have conserved flows. Resulting flow graphs have $|V|_{max}$=20, $|E|_{avg}$≈57.2.

Taxi Trips: From the NYC taxi trips dataset [175], we create *two sets* of flow graphs each corresponding to ***daily passenger flows*** between taxi zones in Manhattan during *8-9am* and *6-7pm* intervals, respectively. We select weekdays in 2017, 2018 and 2019, excluding the summer months. For each weekday, we construct two graphs (one for each set) where the nodes represent taxi zones, a directed edge exist from node u to node v iff there is a trip starting at u and ending at v, and flows ($x_{u,v}$) on edges represent the total number of passengers travelled at a given time interval. Note that some node pairs are likely to have flows in both directions ($x_{u,v} \geq x_{v,u} > 0$). In such cases we only keep the direction with larger flow and update its flow value to the net flow ($x_{u,v} - x_{v,u}$). Lastly, we remove edges with less than 50 flows and take the largest connected component in each graph. This process results in 584 flow graphs for each set, namely ***8-9am*** ($|V|_{max}$=32, $|E|_{avg}$≈133) and ***6-7pm*** ($|V|_{max}$=32, $|E|_{avg}$≈156). Flows are normalized per snapshot.

5.6.2 FlowGEN Training Details

We start with hyper-parameters that we fixed during our experiments as they are empirically found to perform well across all datasets. The tuned hyper-parameters are mentioned at the end of this section.

Our generator ($\mathcal{G}_\theta$) comprises a 4-layer MLP with hidden units of $\{64, 128, 256, N^2\}$, where N is the maximum number of nodes assigned accordingly per each dataset. A *ReLU* activation is applied after each layer. The input to the generator is a 16-dimensional vector drawn from a multivariate standard normal distribution. In the discriminator ($\mathcal{D}_\phi$), we employ four bi-directional message-passing steps. The *in* and *out* message functions ($\{m_i^t, m_o^t\}_{t=1}^4$) are single layer MLPs without activation, where the hidden units are 16 for all steps. Node update functions ($\{f_i^t, f_o^t\}_{t=1}^4$) are two-layer MLPs with hidden units of $\{32, 16\}$, and *ReLU* activations. In the read-out layer, f_{io} is a single layer MLP with 16 hidden units and a *tanh* activation. The context functions f_a and f_b are both single-layer MLPs with hidden units of 16, and the former is followed by a *sigmoid* activation while the latter has no activation. Lastly, the graph-level representation is processed by f_G which is modeled as a 2-layer linear MLP with hidden units of $\{8, 1\}$. Notice that there is no activation for the final layer, and the output is a scalar score $\in \mathbb{R}$ rather than a probability. The model parameters $\{\theta, \phi\}$ are trained using mini-batch SGD with the Adam optimizer [66]. The learning rate is set to 1e-3. L2 regularization is applied for all weights with a penalty of 1e-6. We set the Wasserstein gradient penalty to 10 as suggested by [171] (applied to the discriminator). To allow for a warm start during the training, for the first 10 epochs, we perform four update steps for the parameters of the discriminator for each single update step of the parameters of the generator (*i.e.*, n_critic ratio = 4:1). The hyper-parameters we tune are the n_critic ratio $\in \{4{:}1, 2{:}1, 1{:}1, 1{:}2, 1{:}4\}$ (effective after 10th epoch), the mini-batch size $\in \{32, 64, 128\}$, and the dropout ratio $\in \{0, 0.5\}$ (applied after all MLP layers).

5.6.3 Training Details of Two-Step Approaches

We first detail the training process of each topological graph generative model employed, as well as the modifications performed for adapting them to the problem of generating flow graphs. NetGAN is originally proposed to learn from a single graph. We make the following modifications to NetGAN to make it suitable for our problem setting, where a dataset might have varying graph topologies. We first modify the (ground-truth) random walk sampling process so that the sampled walks take into account the statistics of the entire training set. For that, we construct a meta-graph $\mathbf{G}$, which is represented by the weighted adjacency matrix $\mathbf{A}$=$1/|\mathcal{T}| \sum_{i \in \mathcal{T}} D_i^{-1} A_i$, where A_i and D_i are the adjacency and normalized diagonal node degree matrices of graph G_i, respectively. We then perform random walks on $\mathbf{G}$ such that the transition probability from node u to v is $(\mathbf{D}^{-1}\mathbf{A})_{u,v}$. Second, NetGAN requires an input parameter that decides the number of edges generated. In the case of a single snapshot, this parameter is simply set to the number of edges in the observed graph. However, in order to generate multiple graphs with potentially varying topologies, we instead sample $M_i \sim \mathcal{N}(\mu_{\mathcal{T}}, \sigma_{\mathcal{T}})$ edges for each generated graph G_i, where $\mu_{\mathcal{T}}$ and $\sigma_{\mathcal{T}}$ are the sample mean and standard deviation of the number of edges observed in the entire training set $\mathcal{T}$. No modification is needed for the GRAN baseline. For both models, we set the hyper-parameters based on recommendations made in their respective papers, except for the mini-batch size which is tuned $\in\{32, 64, 128\}$.

Recall from Section 5.4.1 that we employ three approaches as flow generating function $\mathcal{F}$, namely *FMLP*, *FGCN* and *DFF*. The first two extensions predict flows on edges based on topological encoding of participating nodes with the help of neural networks. The latter extension instead assigns flows on edges by solving a flow-balance equation (Equation 5.11). We now detail how these flows are derived, including the training details and hyper-parameter selection whenever applicable.

FMLP and FGCN: As discussed in Section 5.3.1, for both variants, we first generate 16-dim node features $\mathcal{M} \in \mathbb{R}^{|V| \times 16}$ using a standard spectral embedding based on the Laplacian eigenmaps of the input graph's adjacency matrix A. This step allows our two-step variants to handle variable sized graphs as the baseline models are likely to generate graphs with varying number of nodes and edges. Next, we experiment with two different encoder layers: *(1)* The *FMLP* variant consists of two-layer fully connected neural network, *(2)* the *FGCN* variant consists of two graph convolution layers [159]. Both variants apply ReLU activations after each layer. The hidden units are tuned $\in \{8, 16, 32\}$ and mini-batch size is tuned $\in \{32, 64, 128\}$. The loss objective is formally defined as follows:

$$\mathcal{L} = \sum_{A,X \sim P_{data}} \sqrt{\frac{1}{|A_{>0}|} \sum_{(u,v) \in A_{>0}} (\mathcal{F}(A)_{u,v} - X_{u,v})^2} \tag{5.10}$$

where A and X respectively denote the adjacency and flow matrices. $A_{>0} = \{(u, v)|A_{u,v} > 0\}$ represents the non-zero elements of A, while $X_{u,v} \in \mathbb{R}$ is the corresponding flow value directed from node u to v. The parameters of $\mathcal{F}$ are jointly learned by minimizing $\mathcal{L}$ using Adam optimizer with a learning rate of 1e-3. We use the same train/validation/test split as we used for graph generative models for a fair comparison. Early stopping is applied based on the MSE score computed on the validation set with a patience of 10 epochs. Once converged, we then use both variants to predict edge flows on generated graphs by each undirected graph generative model, resulting in four variants: *NetGAN-FMLP, NetGAN-FGCN, GRAN-FMLP* and *GRAN-FGCN.*

Divergence-free flows (DFF): The third variant aims to distribute flow values on the graph's edges so that flows on majority of the nodes are nearly conserved, *i.e.*, total flow that enters a node should be approximately equal to the total flow that leaves. This is a well-studied phenomenon observed in many real-world settings including (but not limited to) transportation networks, water supply networks and power grid networks. For this variant, for each graph $G(V, E)$ generated by undirected graph generative model (*e.g.* GRAN), we first randomly pick 20% of the edges (E^L) and assign them flow values sampled from the ground truth distribution ($\mathbf{X}_{u,v} \sim P_{data}$). Next, ensuring that flows are almost conserved, we compute flow values for the rest of the edges by solving the following constrained optimization problem introduced by [153]:[1]

$$
\begin{aligned}
&X^* = \underset{X}{\operatorname{argmin}} \|(X - X^T)\mathbf{1}\|_2^2 + \lambda \|X\|_F^2 \\
&\text{s.t. } X_{u,v} = \mathbf{X}_{u,v}, \forall (u,v) \in E^L \text{ and } X_{u,v} = 0, \forall (u,v) \notin E
\end{aligned} \tag{5.11}
$$

where $\mathbf{1}$ is a column vector of size $|V|$ whose entries are all 1's. λ is a constant that controls the regularization term —which guarantees a unique optimal solution— and is set to 0.1. Note that X^* may have negative flows after solving the above objective. Therefore, we further apply the following transformation in order to ensure the non-negative flow matrix: $X = max(0, X^* - X^{*T})$.

5.6.4 More Experimental Results

The previous experimental results given in Table 5.1 primarily assess the generated graphs from a flow-centric standpoint, mainly because our objective is to generate realistic flow graphs. That said, an important pillar towards achieving this goal is the ability to reproduce underlying graph topologies found in data. Thus, we present a comprehensive analysis on all four datasets with additional graph statistics in Table 5.2. We also include two more traditional graph generative models as baselines: Erdos-Renyi (E-R) [136] and Barabasi-Albert (B-A) [135]. Results show that FlowGEN consistently yields competitive results against neural baselines while outperforming the traditional approaches.

5.6.5 Stopping Criterion

Notice that all the models employed in our experiments have different underlying objectives. Consequently, an early stopping criterion based on a non-decreasing loss is not

[1]Notation is modified from the original version to be consistent with our setting.

	Simulated Data					**Taxi Trips**				
	Power (top) / **Water** (bottom)					**8-9am** (top) / **6-7pm** (bottom)				
	Deg. dist.	Clus. coeff.	Assort-ativity	△	□	Deg. dist.	Clus. coeff.	Assort-ativity	△	□
test data	–	–	-0.196	16.0	23.0	–	–	-0.255	275.3	2031.4
E-R	0.177	0.114	-8.55e^{-2}	7.92	18.63	8.53e^{-2}	1.99e^{-2}	-7.29e^{-2}	113.4	749.5
B-A	0.479	0.998	-0.428	0.0	0.0	6.80e^{-2}	2.02e^{-2}	-0.195	120.2	771.2
NetGAN	5.32e^{-2}	0.172	-0.247	7.95	12.19	3.54e^{-2}	2.03e^{-2}	-0.223	87.3	416.7
GRAN	**3.86e^{-3}**	**4.27e^{-2}**	**-0.188**	**16.65**	25.60	**5.32e^{-3}**	**1.97e^{-2}**	**-0.251**	231.9	1768.8
FlowGEN	1.91e^{-2}	0.106	-0.221	13.20	**21.23**	1.48e^{-2}	1.99e^{-2}	-0.234	**256.4**	**2170.2**
test data	–	–	-2.63e^{-2}	36.69	108.84	–	–	-7.86e^{-2}	340.3	2523.0
E-R	2.48e^{-2}	2.51e^{-2}	-0.131	17.5	57.96	9.21e^{-2}	2.01e^{-2}	-6.36e^{-2}	173.1	1277.0
B-A	0.101	4.62e^{-2}	-0.322	13.32	38.78	6.11e^{-2}	2.00e^{-2}	-0.186	197.4	1460.3
NetGAN	2.59e^{-2}	2.23e^{-2}	-0.138	22.2	56.4	6.20e^{-2}	2.02e^{-2}	**-7.23e^{-2}**	94.5	441.4
GRAN	**4.20e^{-3}**	**2.08e^{-2}**	-5.54e^{-2}	38.37	116.39	**3.01e^{-3}**	2.00e^{-2}	-0.152	**358.5**	2876.4
FlowGEN	9.27e^{-3}	2.19e^{-2}	**-3.49e^{-2}**	**36.01**	**102.38**	1.57e^{-2}	**1.99e^{-2}**	-6.89e^{-2}	299.7	**2671.3**

Table 5.2: Results with respect to *graph topology*. Metrics from left: (1) degree and (2) clustering coefficient distributions; average (3) degree assortativity and (4-5) number of 3&4-cycles. For MMD scores (1-2), the smaller the better; for average statistics (3-5), the closer to test data, the better.

appropriate and is likely to result in unfair comparisons. In this work, we instead apply an early stopping criterion that is based on the model performance measured with respect to evaluation metrics. Since we employ a collection of metrics that measure the performance from various perspectives (recall from Section 5.4.1), considering only a single metric (*e.g.* degree distribution) might not translate to similar performance in other metrics (*e.g.* divergence distribution) due to the complex nature of flow graphs. Therefore, we continue training the models—evaluating after each epoch—until more than half of the metrics stop improving with a patience of 10 epochs. For the baseline models, we re-sample the flows before each evaluation step as the generated graphs are likely to change. Once the training is complete, we rank all the snapshots based on the evaluation metrics and report the best-ranked snapshot for each model.

5.6.6 Case Study: European Power Network

We perform a case study focused on the generation of a particular type of flow graphs, namely power networks. They are often applied in the analysis of large power distribution infrastructure, including their robustness [176] and optimization [177]. Moreover, the dynamics of power networks is governed by physical laws—in this case, Kirchhoff's laws—posing a great challenge to domain-agnostic generative models for flow graphs. In this case study, we apply a new domain-specific evaluation metric in order to assess whether the generated graphs are consistent with the expected properties of power networks.

Evaluation: Besides the metrics described in Section 5.4.1, we also apply a more specific one to evaluate power flow graphs. Our goal is to assess whether the generated flows are consistent with the dynamics observed in real power networks. For each generated flow graph $G(V, E, X)$, we compute node divergences $X_u^{div} = (X_u^{out} - X_u^{in})$, where X_u^{in} and X_u^{out} are total in and out flows for node u, respectively. These values capture the net power consumed ($X_u^{div} < 0$) or generated ($X_u^{div} > 0$) at each node in the flow graph. However, notice that different assignments of flows can satisfy the same node divergences—flows can be increased arbitrarily along the cycles in G—and thus one can define a notion of optimality to compare them. We define the optimal flow assignment X^* given node divergences X^{div} and a set of edges E using a formulation similar to Equation 5.11:

$$\begin{aligned} X^* &= \underset{X}{\operatorname{argmin}} \, \|(X - X^T)\mathbf{1} - X^{div}\|_2^2 + \lambda\|X\|_F^2 \\ &\text{s.t. } \forall X_{u,v} > 0, (u, v) \in E \end{aligned} \tag{5.12}$$

where λ (set to 0.1) is a regularization parameter. X^* attempts to satisfy the node divergences as those in X^{div} while also minimizing a regularization term. Besides making the solution unique, this regularization has a physical interpretation from Kirchhoff's

	Eff. score
test data	0.69
NetGAN-FGCN	0.47
GRAN-FGCN	0.52
FlowGEN	**0.58**

Table 5.3: Power network generation results with respect to the efficiency score. The higher the score, the better.

NetGAN	GRAN	FlowGEN
0.176	1.293	0.154

Table 5.4: Runtime comparisons of models (in seconds).

voltage law [178]. Intuitively, it penalizes cyclic flows, which are not driven by differences of potentials, while smoothly distributing flow values as if line resistances are uniform. Therefore, given the same node divergences and graph topology, one would expect an effective model to produce flows that are close to X^*. We define the *efficiency score* of a flow matrix X as $1 - ||X - X^*||_F$, where both flow matrices are normalized.

Experimental Results: Table 5.3 shows the performance of FlowGEN and the best performing baseline variants on the power dataset. In Section 5.4.2 we showed that while GRAN variants excel at reproducing topological properties, FlowGEN achieves the best results regarding flow-related metrics (*i.e.*, edge flows, node divergences and directed cycles) at both local and global levels. Similarly, regarding the efficiency score, results show that FlowGEN produces flow graphs that are closer to optimal compared to the baselines. As expected, the test graphs have the highest efficiency score. Moreover, notice that neither FlowGEN nor the baselines take into account the objective function from Equation 5.12 explicitly. These results give substantial evidence that FlowGEN is able to capture complex domain-specific flow dynamics observed in the data.

5.6.7 Runtimes

Due to our rather complicated early stopping criterion, it is not reasonable to compare total training times of models. Since baseline models are not designed to generate flow graphs, they tend to have a fluctuating performance on flow-related metrics, which often results in longer training times. Instead, we report one full pass (single iteration) time for each model on *Taxi Trips 8-9am* in Table 5.4. Runtimes are measured with batch size of 32 on a GeForce GTX 1070. As we can see, NetGAN and FlowGEN have similar runtimes, while GRAN is significantly slower due to its sequential generation process.

5.6.8 Ablation Study

In this section, we conduct an ablation study in order to demonstrate the contributions of each FlowGEN component by disabling these components one by one, and replacing them with standard approaches from the literature. We name these variants as follows:

- FlowGEN without Flow-Pooling Layer (*w/o FPL*): This variant disables our flow-pooling component and instead uses Gaussian random vectors as initial *in* and *out* node features ($[{}_{in}h_u^1; {}_{out}h_u^1] \sim \mathcal{N}(0, I_6)$). This is a well-known approach for cases where external node features are not available [161].

- FlowGEN without Bidirectional Message-Passing Layer (*w/o BMPL*): This variant disables our bidirectional message passing scheme where the node state updates are computed using two independent (*in* and *out*) neural message-passing channels (Equations 5.6 and 5.7). We instead employ a single message-passing channel to simultaneously update both *in* and *out* node representations. More formally, for each node u at layer t, the respective node update is computed as follows:

$$
\begin{aligned}
&h_u^{t+1} = f^t(h_u^t, m_u^t), \quad t = 1,\cdots,T{-}1\\
&\text{where}\\
&m_u^t = \sum_{v \in G_{in}(u)} m^t(h_v^t, x_{v,u}) + \sum_{v \in G_{out}(u)} m^t(h_v^t, x_{u,v})\\
&h_u^1 = [{}_{in}h_u^1 ; {}_{out}h_u^1]
\end{aligned}
$$

$\{f^t, m^t | t = 1, \cdots, T{-}1\}$ are again modeled as MLPs.

- FlowGEN without Attention-based Readout Layer (*w/o ARL*): This variant replaces the attention-based readout layer (also called gated sum [168]) with a standard SUM aggregator. More formally, we modify Equation 5.9 and compute the latent graph representation as $h_G = \frac{1}{|V|}\sum_{u \in V} \tilde{h}_u$ which simply assigns equal weights to all nodes in the graph.

We also tune hyper-parameters for all variants in a same way as we do for FlowGEN, which is described in Section 5.6.2.

Analysis: Table 5.5 demonstrates that each FlowGEN component contributes to final model performance, where different components have varying effects on different evaluation metrics. We observe that the proposed bidirectional message-passing layer (*BMPL*) enables our discriminator to better capture *higher-order directed* flow characteristics found in ground-truth data, which results in significant improvements on related metrics such as in/out degree distributions, node-divergence distribution and directed k-cycles. With that observation, we hypothesize that bidirectional message-passing channels can account for heterogeneous in-flow and out-flow dynamics of nodes. As an example, for certain nodes like divergence-free nodes, larger total in-flow naturally indicates larger total out-flow, while this does not hold for some other nodes like *sink* or *source* nodes, for which the

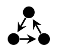
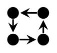
	In-deg. dist.	Out-deg. dist.	$\{x_{u,v}\}$	$\{X_u^{div}\}$		
test data	–	–	–	–	2.58	2.33
FlowGEN	$1.35e^{-2}$	$9.36e^{-3}$	$3.73e^{-3}$	$2.98e^{-3}$	2.66	2.41
w/o FPL	$2.51e^{-2}$	$3.17e^{-2}$	$4.19e^{-3}$	$9.23e^{-3}$	2.60	2.56
w/o BMPL	$4.25e^{-2}$	$8.63e^{-2}$	$4.39e^{-3}$	$5.73e^{-2}$	0.85	1.48
w/o ARL	$1.81e^{-2}$	$2.38e^{-2}$	$3.08e^{-3}$	$7.16e^{-3}$	1.83	3.64

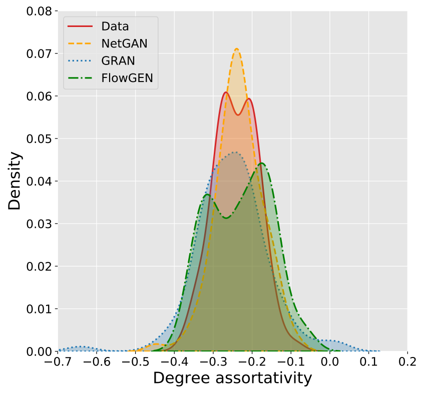
Table 5.5: Ablation study of various model components on Power dataset. Metrics from left to right: Distributions of (1-2) in&out degree, (3) edge flow, and (4) node divergence; (5-6) average number of directed 3&4-cycles. For MMD scores (1-4), the smaller the better; for average statistics (5-6), the closer to test data, the better.

relationship between the total incoming and outgoing flow is clearly not linear.

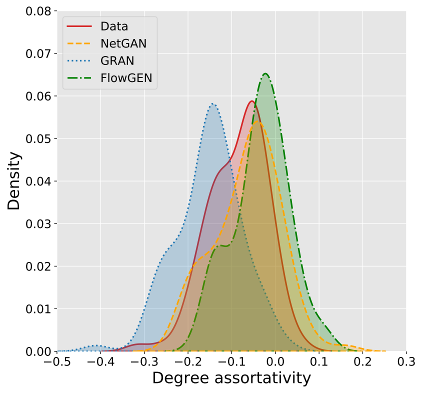
Moreover, while the flow-pooling layer (*FPL*) does not seem to have a notable impact on directed cyclic trends, it leads to improvements on in/out degree distributions and (more prominently) on node-divergence distribution. This suggests that our flow-pooling operators provide better inductive bias towards discerning node-level flow dynamics. In other words, we observe that explicitly assigning rather simple flow statistics of nodes as initial node states—in addition to inducing the key property of permutation invariance—results in improved performance in capturing global node-level flow characteristics, as well as directed connectivity patterns found in data. Another interesting observation we have is that *without FPL*, our model is often more susceptible to *mode collapse* and the stopping criterion is triggered at earlier epochs.

The third component of our discriminator is the attention-based readout layer (*ARL*). Such a layer is widely used in the literature to adjust the contributions of each node to the final (graph-level) task of interest, as intuitively, certain nodes play a more critical role than others [179]. In this work, we leverage this technique to improve our model's ability to replicate critical nodes in flow graphs (such as *source* and *sink* nodes), which directly leads to desirable improvements observed through both qualitative (Figure 5.3) and quantitative (Table 5.5) evaluations. More specifically, we observe that employing an attention-based readout layer leads to enhanced performance on the node-divergence distribution and directed k-cycle count metrics compared to the standard SUM aggregator, which assigns equal weights to all nodes in the graph.

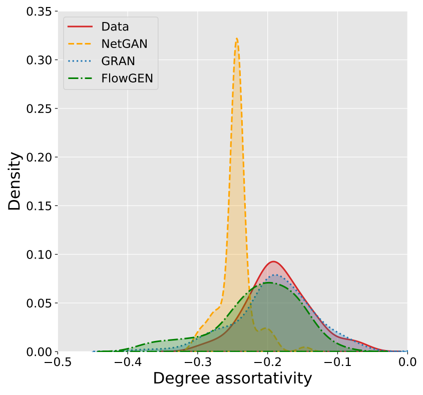
The results discussed in this section confirm the different choices made in the design of the FlowGEN architecture. As discussed above, each component contributes to our model's ability to reproduce one or more desired properties of real flow graphs, which results in desired performance when most of the metrics considered (see Table 5.1).



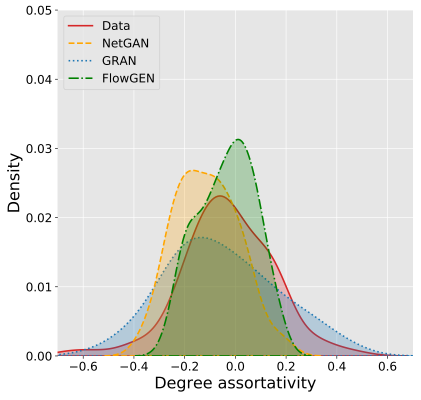
(a) Taxi 8-9am



(b) Taxi 6-7pm



(c) Power



(d) Water

Figure 5.5: Distributions of degree assortativity coefficients observed in graphs from training data, NetGAN, GRAN, and FlowGEN for all four datasets used in our experiments.

5.6.9 Diversity Analysis

Lastly, we analyze the diversity of our generated graphs, which is an important aspect in assessing generative models' capacity as they are known to suffer from mode collapse [170, 180]. In the context of graph generative models, the notion of diversity is typically defined for undirected graphs and does not directly apply to our case (e.g. graphs with the same topology but different flow distributions are considered different). That said, we have observed that FlowGEN is able to generate diverse sets of flow graphs with varying topological and flow-related characteristics.

Figure 5.5 demonstrates the distribution of generated graph structures observed in all four datasets, as well as graphs generated by NetGAN, GRAN, and FlowGEN. To do that, we choose (undirected) degree assortativity coefficient [181] as our metric, which is a graph-level scalar value allowing us to conveniently examine and compare structural diversity across generated graphs. For FlowGEN, we use the undirected versions of the generated flow graphs for a fair comparison. Our results show that while FlowGEN and GRAN generate diverse underlying graph structures, NetGAN tends to generate graphs with similar structures, particularly for Taxi 8-9am (Figure 5.5a) and Power (Figure 5.5c).

Chapter 6

Problems on Network Regularization

In this chapter, we introduce our work on two problems that can be grouped under the umbrella term of *network regularization*. Network regularization has proven as an effective tool for learning more robust models that can better generalize to unseen data, at the same time making more sample efficient use of the available training data. Moreover, it allows us to take advantage of prior knowledge on network structures and capture relational inductive bias that is rooted in data.

Section 6.1 of this chapter introduces a generic network regularization framework, which facilitates more robust learning in the cases of missing and/or corrupt edge weights, a setting that is commonly encountered in real-world problems. Section 6.2 studies the challenging problem of missing flow estimation on flow graphs. It introduces a novel physics-informed network regularization technique that can jointly account for network structure, node/edge features, and the underlying domain physics. Through extensive experiments using large-scale road traffic and electric power networks, we show that the proposed framework not only achieves superior performance compared to the baselines, but also learns interpretable physical properties, such as the role played by resistance in a power transmission network and by the number of lanes in a road traffic network.

6.1 Discrepancy-aware Network Regularization

6.1.1 Introduction

Network regularization is a fundamental approach to encode and incorporate general relationships among variables, which are represented as nodes and linked together by weighted edges that describe their local proximity. A significant amount of effort has been devoted to developing successful regularizations [182, 183] that take advantage of prior knowledge on network structures to enhance estimation performance for a variety of application settings, including image denoising [184] and genomic data analysis [185].

We consider the general setting of network regularization, now proposed as a convex optimization problem defined on an undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with node set $\mathcal{V}$ and edge set $\mathcal{E}$:

$$\min \sum_{i \in \mathcal{V}} f_i(x_i) + \lambda \sum_{(j,k) \in \mathcal{E}} \omega_{jk} \cdot g(x_j, x_k), \tag{6.1}$$

where on each node i, we intend to learn a local model $x_i \in \mathbb{R}^d$, which in addition to minimizing a predefined *convex* loss function $f_i : \mathbb{R}^d \to \mathbb{R}$, carries certain relations to its neighbors, defined by the function $g : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$.

In this work, we focus on the particular setting where $g(x_i, x_j) = \|x_i - x_j\|_2$, also known as a *sum-of-norms* (SON) regularizer. It assumes that the network is composed of multiple clusters, suggesting that all nodes within a cluster share the same consensus model ($x_i = \cdots = x_j$). As the observed data on each node may be sparse, the SON regularizer allows nodes to "borrow" observations from their neighbors to improve their own models, as well as to determine the network cluster to which they belong. The SON objective was first introduced in [186] for convex clustering problems and used off-the-shelf sequential convex solvers. Chi & Lange [187] adapt SON clustering to incorporate similarity weights with an arbitrary norm and solve the problem by both alternating direction method of multipliers (ADMM) or alternating minimization algorithm (AMA) in a parallel manner. More recently, Hallac et al. [182] point out that weighted SON (named as Network Lasso) allows for simultaneous clustering and optimization on graphs, and is highly suitable for a broad class of large-scale network problems.

In network regularization, a static weight assigned to an edge determines how strictly the difference between models on the corresponding nodes is being penalized, relative to the other node pairs in the network. However, unlike few scenarios in which network information is explicitly given, edge weights are usually unavailable or even infeasible to obtain in most real-world networks (e.g., gene regulatory networks [188]). Moreover, due to possible measurement errors and inaccurate prior knowledge, the assigned edge weights don't necessarily align with the underlying clustering structure of networks as shown in [189, 190]. As a result, heavily relying on the edge weights in determining how much penalty should be applied to neighboring models may contaminate the discovered solutions.

Similar intuition applies when we extend to the temporal setting. Models on nodes of temporal networks usually change at the level of groups over time. However, some groups exhibit different evolution patterns than others [191]. Moreover, the grouping structures themselves may evolve with time [192]. As static regularization cannot capture such temporal evolution, a direct solution is to induce a time-varying local consensus by employing the SON objective on both spatial and temporal directions, but we face an more drastic problem: how to assign weights between snapshots.

Consequently, a framework that is robust to missing or corrupted edge weights in network-based regularization and adapts to spatio-temporal settings is desired. Note that without the assigned network weights, the setting in Eq. (6.1) degrades to simply applying isotropic regularization for all the edges. In this work, we propose a generic formulation, called the *discrepancy-aware network regularization* (DANR), which deploys a suitable amount of anisotropic network regularization in both spatial and temporal aspects. DANR infers models at each node per timestamp and can learn evolution of models and transitions of network structures over time. We develop an ADMM-based algorithm that adopts an efficient and distributed iterative scheme to solve problems formulated by the DANR, and show that the proposed solution obtains guaranteed convergence towards global optimal solutions. By applying to both synthetic and real-world datasets, we demonstrate the effectiveness of the proposed approach on various network problems.

6.1.2 Related Work

Among existing network-based regularization approaches, there are two major types of edge objectives that are most commonly used: the square-norm objective and the unsquared-norm objective, encouraging different styles of expected solutions. The squared-norm edge objective (i.e., $\sum_{jk} \omega_{jk} \|x_j - x_k\|_2^2$), well-known as *graph Laplacian regularization* [193], assumes underlying models on nodes are smoothly varying as one traverses edges in the network, and accordingly enforces similar but not identical models on linked nodes. Due to the merits of simple computations and good performance, graph Laplacian regularizer has gained popularity in solving problems, such as image deblurring and denoising [184], genomic data analysis [185], semi-supervised regression [194], non-negative matrix factorization [195] and semi-definite programming [196]. However, the square-norm objective usually induces very dense models. The unsquared-norm objective, known as *sum-of-norms* regularizer [186, 197], bears some similarity to *scale-valued fused lasso* signal approximator [198] that exists in signal processing applications, for example, *total-variation* (TV) regularizer for image denoising, and *graph trend filtering* (GTF) regularizer for nonparametric regression. TV regularizer [199] is developed to penalize both the horizontal and vertical differences between neighboring pixels, which can be thought of as a special scalar-valued network lasso on a 2D grid network. GTF [183] generalizes the successful idea of trend filtering to graphs, by directly penalizing higher order differences across nodes. All regularizers above require correct edge weights so that network structures can be properly used to improve joint estimation. In contrast, our proposed approach allows ambiguous input of network edge weights by introducing the discrepancy-buffering variables. Also, as a variation of SON regularization, our formulation enjoys small model complexity as local consensus is imposed across large-scale networks.

6.1.3 Problem Setting

Since existing network-based regularizers rely on known weights on edges, the corresponding solutions get misled when the edge weights are erroneous or unknown. Directly learning the unknown edge weights by using the same regularizer as Eq. (6.1) would lead to an optimization problem:

$$\min_{x,\omega} \quad \sum_{i\in\mathcal{V}} f_i(x_i) + \lambda \sum_{(j,k)\in\mathcal{E}} \omega_{jk} \left\|x_j - x_k\right\|_2 \tag{6.2}$$

which yields the trivial all-zero solution of ω and thus becomes unsatisfactory. Instead of imposing more sophisticated model-based or problem-dependent regularization as suggested in [200], we consider an alternative formulation that explicitly accounts for discrepancies between models on adjacent nodes:

$$\begin{aligned} &\min_{x,\alpha} \quad \sum_{i\in\mathcal{V}} f_i(x_i) + \lambda \cdot \mathcal{R}_S(x,\alpha) \\ &\mathcal{R}_S(x,\alpha) = \mu \sum_{(j,k)\in\mathcal{E}} \omega_{jk} \left\|x_j + \alpha_{jk} - x_k\right\|_2 + (1-\mu)\|\alpha\|_{1,p} \end{aligned} \tag{6.3}$$

where we denote $\|\alpha\|_{1,p} = \sum_{(j,k)\in\mathcal{E}} \|\alpha_{jk}\|_p$ as the sum of p-norms of all α_{jk}'s. We set a penalty parameter $\lambda \geq 0$ to control the overall strength of network regularization, and a portion parameter $0 < \mu < 1$ to control the emphasis between the two terms in $\mathcal{R}_S(x, \alpha)$. We name this formulation in Eq. (6.3) the *discrepancy-aware network regularization* (DANR).

In the first term of $\mathcal{R}_S$, we define a *discrepancy-buffering (DB) variable* $\alpha_{jk} \in \mathbb{R}^d$ to denote the preserved discrepancy between local model parameters x_i and x_j. More specifically, when an edge weight ω_{jk} is given but potentially imprecise or otherwise corrupted, α_{jk} would compensate for abnormally large differences between models x_j and x_k to reduce the magnitude of ω_{jk}-weighted edge penalty term in $\mathcal{R}_S$. The DB variable provides additional flexibility to its associated models, allowing them to stay adequately close to their own local solutions w.r.t. minimizing local loss functions, and avoid over-penalized consensus. When all ω_{jk}'s are not given, α_{jk}'s enable solving Eq. 6.3 under anisotropic regularizations even with homogeneous weights.

The second term of $\mathcal{R}_S$ is the $\ell_{1,p}$-regularizer (with $1 < p < \infty$) [201] that ensures sparsity at the group level. Note that we regard each variable vector α_{jk} as a group ($|\mathcal{E}|$ groups in total). The first key intuition here is that the regularization on α helps to exclude trivial solutions, that is $\alpha_{jk} = x_k - x_j$. Second, it allows us to identify a succinct set of non-zero vectors α_{jk}'s, compensating for possible intrinsic discrepancies between two adjacent nodes.

To sum up, discrepancy-buffering variable α_{jk}'s are designed to elaborately adjust network regularization strength on all edges, and thus reduce negative effects of the unsquared norm regularizer. We will show in later sections that this modified formulation remains convex and tractable via parallel optimization algorithms on large-scale networks.

6.1.4 Distributed ADMM-based Solution

In this section, we propose an ADMM-based algorithm for solving the proposed problem in Eq. (6.3) and present the convergence and complexity of the proposed algorithm. Extensions of our solution to the spatio-temporal case is included in the following section.

The ADMM method was originally derived in [202] and has been reformulated in many contexts including optimal control and image processing [203]. The method can be considered as combining augmented Lagrangian methods and the method of multipliers [204]. It aims to solve optimization problems with two-block separable convex objectives in the following form of

$$\min \; f(x) + g(z) \qquad s.t. \; Ax + Bz = c \tag{6.4}$$

Observe that our proposed objective in Eq. (6.3) has a separable convex objective function: it can be reorganized into two-block separable convex objectives as in Eq. (6.4), for which ADMM methods guarantee convergence to global optimal solutions [205].

More precisely, to fit our problem into the ADMM framework, we first define $x{=}\{x_j\}^{j\in V}$ for model parameter vectors on the given undirected network $\mathcal{G}$, and define consensus variables $u = [\{u_{jk}, u_{kj}\}^{(j,k)\in\mathcal{E}}]$ as copies of x in the spatial penalty term $\mathcal{R}_S(x, \alpha)$. Then we rewrite Eq. (6.3) as follows:

$$\begin{aligned} \min_{x,u,\alpha} \quad & \sum_{j\in\mathcal{V}} f_j(x_j) + \lambda_1(1-\mu_1)\|\alpha\|_{1,p} + \lambda_1\mu_1 \sum_{(j,k)\in\mathcal{E}} \omega_{jk} \left\|u_{jk} + \alpha_{jk} - u_{kj}\right\|_2 \\ \text{s.t.} \quad & [u_{jk}, u_{kj}] = [x_j, x_k]\,, \quad (j,k)\in\mathcal{E} \end{aligned} \tag{6.5}$$

in which the first term corresponds to the f block in Eq. (6.4), and the remaining terms correspond to the g block. The equality constraints on Eq. (6.3) are used to force consensus between variables x and u. Next, we derive the augmented Lagrangian of Eq. (6.5):

$$\mathcal{L}_{\rho_1}(x,u,\alpha,\delta) = \sum_{j\in V} f_j(x_j) + \lambda_1(1-\mu_1)\|\alpha\|_{1,p} + \sum_{(j,k)\in E}\Big(\lambda_1\mu_1\omega_{j,k}\|u_{jk}+\alpha_{jk}-u_{kj}\|_2 \\ + \frac{\rho_1}{2}\|x_j-u_{jk}+\delta^u_{jk}\|_2^2 - \frac{\rho_1}{2}\|\delta^u_{jk}\|_2^2 + \frac{\rho_1}{2}\|x_k-u_{kj}+\delta^u_{kj}\|_2^2 - \frac{\rho_1}{2}\|\delta^u_{kj}\|_2^2\Big) \tag{6.6}$$

where $\delta = [\{\delta^u_{jk}, \delta^u_{kj}\}^{(j,k)\in\mathcal{E}}]$ are scaled dual variables for each equality constraint on the elements of u. The parameters $\rho_1 > 0$ penalize the violation of equality constraints in the spatial and temporal domain [206] respectively. The iterative scheme of ADMM under the above setting can be written as follows, with l denoting the iteration index:

$$\left.\begin{aligned} x^{(l+1)} &= \underset{x}{\operatorname{argmin}}\ \mathcal{L}_{\rho_1}(x,(u,\alpha,\delta)^{(l)}) \\ (u,\alpha)^{(l+1)} &= \underset{u,\alpha}{\operatorname{argmin}}\ \mathcal{L}_{\rho_1}(x^{(l+1)},u,\alpha,\delta^{(l)}) \\ \delta^{(l+1)} &= \delta^{(l)} + (\tilde{x}^{(l+1)} - u^{(l+1)}) \end{aligned}\right\} \tag{6.7}$$

where $\tilde{x}$=$[\{x_j, x_k\}^{(j,k)\in\mathcal{E}}]$ is composed of replicated elements in x, and thus has a one-to-one correspondence with elements in u.

Because the augmented Lagrangian in Eq. (6.6) has a separable structure as well, we can further split the optimization above over each univariate element in x and z. Next, we provide details for each ADMM update step.

x-Update. In the first update step of our ADMM updating scheme, we can decompose the problem of minimizing x into separately minimizing x_j for each node j :

$$x_j^{(l+1)} = \underset{x_j}{\operatorname{argmin}}\Big(f_j(x_j) + \frac{\rho_1}{2}\sum_{k\in N_j}\|x_j - u^{(l)}_{jk} + \delta^{u(l)}_{jk}\|_2^2\Big) \tag{6.8}$$

As above equation suggests, x_j on node j first receives local information from the corresponding consensus variables belonging to all of its neighbors $k \in N_j$, then updates its value to minimize the loss function and remain close to neighbouring consensus variables. Since all remaining regularizations are quadratic, the x-update problem can be efficiently solved whenever the loss function f_j has certain properties, such as strong convexity.

(u, α)-Update. We can further decompose the (z, α)-update step in Eq. (6.7) into subproblems on each edge (and its related variables $u_{jk}, u_{kj}, \alpha_{jk}$) as follows, which can be solved in parallel:

$$\begin{aligned}(u_{jk}^{(l+1)}, u_{kj}^{(l+1)}, \alpha_{jk}^{(l+1)}) = \text{argmin } \Big(&\lambda_1(1-\mu_1)\|\alpha_{jk}\|_p + \lambda_1\mu_1\omega_{jk}\|u_{jk} + \alpha_{jk} - u_{kj}\|_2 \\ &+ \frac{\rho_1}{2}\|x_j^{(l+1)} - u_{jk} + \delta_{jk}^{u(l)}\|_2^2 + \frac{\rho_1}{2}\|x_k^{(l+1)} - u_{kj} + \delta_{kj}^{u(l)}\|_2^2\Big) \qquad (6.9)\end{aligned}$$

Notice that the sum of convex functions which are defined on different sets of variables preserves the convexity. To minimize the objective with u_{jk}, u_{kj} and α_{jk} simultaneously, the convexity of Eq. (6.9) motivates us to adopt an alternating descent algorithm, which minimizes each component iteratively with respect to (u_{jk}, u_{kj}) and α_{jk} while holding the other fixed. In detail, we solve Eq. (6.10) and Eq. (6.11) iteratively until convergence is achieved:

$$\begin{aligned}(u_{jk}^{(l'+1)}, u_{kj}^{(l'+1)}) &= \text{argmin}\Big(\lambda_1\mu_1\omega_{jk}\|u_{jk} + \alpha_{jk}^{(l')} - u_{kj}\|_2 \qquad (6.10)\\ &\quad + \frac{\rho_1}{2}\|x_j^{(l+1)} - u_{jk} + \delta_{jk}^{u(l)}\|_2^2 + \frac{\rho_1}{2}\|x_k^{(l+1)} - u_{kj} + \delta_{kj}^{u(l)}\|_2^2\Big)\\ \alpha_{jk}^{(l'+1)} &= \text{argmin}\Big((1-\mu_1)\|\alpha_{jk}\|_p + \mu_1\omega_{jk}\|u_{jk}^{(l'+1)} + \alpha_{jk} - u_{kj}^{(l'+1)}\|_2\Big) \qquad (6.11)\end{aligned}$$

where l' denoting the iteration index of alternating descent. Further, we can utilize the analytic solution to speed up the calculation of u_{jk} and u_{kj}:

Lemma 6.1.1 *Problem (6.10) has a closed-form solution:*

$$\left.\begin{aligned} u_{jk}^{(l'+1)} &= (1-\theta)a + \theta b - \theta\alpha_{jk}^{(l')} \\ u_{kj}^{(l'+1)} &= \theta a + (1-\theta)b + \theta\alpha_{jk}^{(l')} \end{aligned}\right\}$$

where we denote $a = x_j^{(l'+1)} + \delta_{jk}^{u\,(l')}$, $b = x_k^{(l'+1)} + \delta_{kj}^{u\,(l')}$, $c = \lambda_1(1-\mu_1)\omega_{jk}$, *and* $\theta = c/(\rho_1\|a - b + \alpha_{jk}^{(l')}\|_2)$ *for simplification. Details are in [4].*

δ-Update. In the last step of our ADMM updating scheme, we have fully independent update rules for each scaled dual variable δ_{jk}^u as follows:

$$\delta_{jk}^{u\,(l+1)} = \delta_{jk}^{u\,(l)} + (x_j^{(l+1)} - u_{jk}^{(l+1)}) \tag{6.12}$$

Stopping Criterion and Global Convergence. For our ADMM iterative scheme in Eq. (6.7), we use the norm of primal residual $r^{(l)} = \tilde{x}^{(l)} - u^{(l)}$ and dual residual $s^{(l)} = \rho_1(u^{(l)} - u^{(l+1)})$ as the termination measure. The optimality condition [204] of ADMM shows that if both residuals are small then the objective suboptimality must be small, and thus suggests $\|r^{(l)}\|_2 \le \epsilon_1 \wedge \|s^{(l)}\|_2 \le \epsilon_2$ as a reasonable stopping criterion. Convex subproblems in x-update and (u, α)-update need iterative methods to solve as well. The stopping criterion for these subproblems is naturally to keep iteration differences below thresholds, *i.e.* in (u, α)-update, we require $\|\Delta u^{(l')}\|_2 \le \epsilon'$ and $\|\Delta\alpha^{(l')}\|_2 \le \epsilon'$.

Lemma 6.1.2 *Our ADMM approach to solve the DANR problem is guaranteed to converge to the global optimum. Proof is in [4].*

Computational Complexity. Let N_c denote the number of iterations that ADMM takes to achieve an approximate solution $\hat{x}$ with an accuracy of ϵ_x. Based on the convergence analysis in [207], the time complexity scales as $O(1/\epsilon_x)$, which in our problem mostly depends on the properties of cost functions f_{jt}'s. Assume that all convex subproblems in x-update and (u, α)-update are solved by general first-order gradient descent methods that take N_x and N_α iterations to converge respectively, the overall complexity of the algorithm is therefore $O\big(N_c\big(N_x|\mathcal{V}| + N_\alpha|\mathcal{E}| + |\mathcal{E}|\big)\big)$.

6.1.5 Extension to Spatio-temporal Setting

The aforementioned shortcomings with pre-defined edge weights accentuate when we consider temporal networks since acquiring explicit temporal edge weights is usually not feasible. Thus, discrepancy-buffering variables are also crucial for the temporal setting.

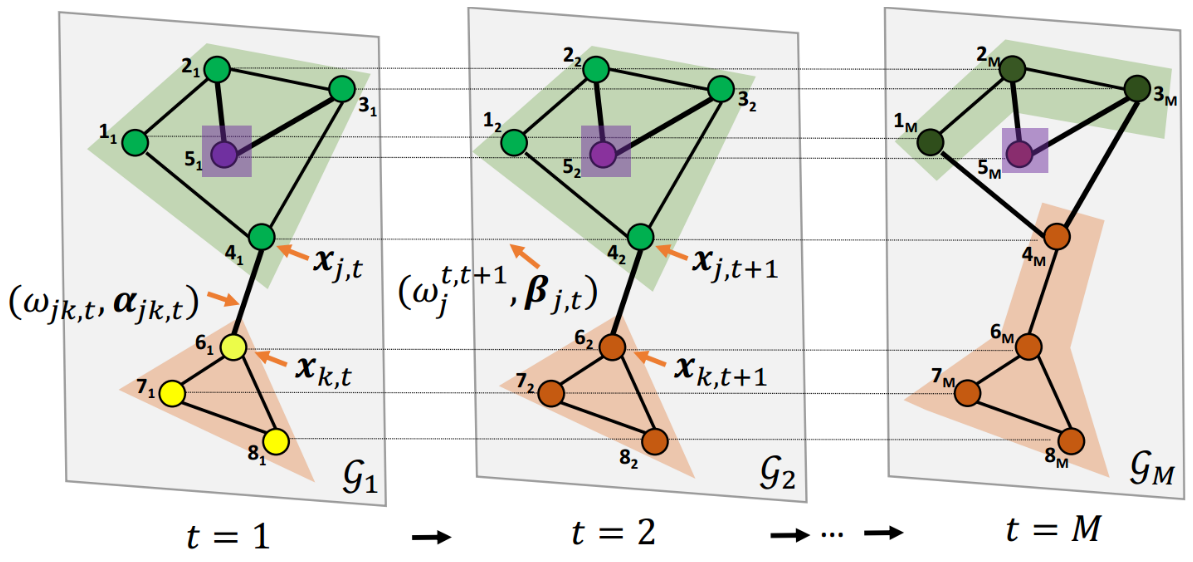


Figure 6.1: Overview of our problem setting in temporal networks.

Consider a temporal undirected network $\mathcal{G}$ consisting of M sequential network snapshots $\{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_M\}$, where each network snapshot $\mathcal{G}_t$ contains a node set $\mathcal{V}_t$ and an edge set $\mathcal{E}_t$. We denote $\omega_{j,k}^t$ as the weight of spatial edge (j_t, k_t) between nodes j_t and k_t within snapshot $\mathcal{G}_t$, and $\omega_j^{t,t+1}$ for the weight of temporal edge (j_t, j_{t+1}) that links node j_t with j_{t+1} across snapshots $\mathcal{G}_t$ and $\mathcal{G}_{t+1}$ (shown in Fig.6.1). On each node j_t, a *convex* loss function $f_{j,t} : \mathbb{R}^d \to \mathbb{R}$ is given to measure the fitness of a local model parameterized by $x_{j,t} \in \mathbb{R}^d$. With sparse observations for all snapshots, a straightforward task is to find jointly optimal models for every node and every timestamp under the regularization for both network topology and temporal evolution. We can propose an extension of the DANR formulation to spatio-temporal networks, referred as **ST-DANR**:

$$\min_{x,\alpha,\beta} \quad \sum_{j\in\mathcal{V}} \sum_{t=1}^{M} f_{j,t}(x_{j,t}) + \lambda_1 \cdot \mathcal{R}_S(x,\alpha) + \lambda_2 \cdot \mathcal{R}_T(x,\beta) \tag{6.13}$$

where we define the discrepancy-aware regularizers as:

$$\begin{aligned}\mathcal{R}_S(x,\alpha) &= \mu_1 \sum_{t=1}^{M} \sum_{(j_t,k_t)\in\mathcal{E}_t} \left(\omega_{jk}^{t} \left\|x_{j,t} + \alpha_{jk,t} - x_{k,t}\right\|_2\right) \\ &\quad + (1-\mu_1)\|\alpha\|_{1,p} \qquad \text{(spatial penalty term)} \qquad (6.14)\\ \mathcal{R}_T(x,\beta) &= \mu_2 \sum_{j\in\mathcal{V}} \sum_{t=1}^{M-1} \left(\omega_{j}^{t,t+1} \|x_{j,t} + \beta_{j,t} - x_{j,t+1}\|_2\right) \\ &\quad + (1-\mu_2)\|\beta\|_{1,p} \quad \text{(temporal penalty term)} \qquad (6.15)\end{aligned}$$

As we are interested in the heterogeneous evolution of nodal models across time, the chosen unsquared norms in spatial and temporal regularization terms $\mathcal{R}_S(x,\alpha)$ and $\mathcal{R}_T(x,\beta)$ enforce piecewise consensus in both spatial and temporal aspects, which indicates abrupt changes of regional models or sudden transitions in network structures at particular timestamps, and also implies valid persistent models in the remaining segments of time [208]. Beside the spatial DB variables $\alpha_{jk,t}$ in $\mathcal{R}_S(x,\alpha)$, we define another set of temporal DB variables $\beta_{j,t} \in \mathbb{R}^d$ in $\mathcal{R}_T(x,\beta)$ to compensate for inadequate regularizations caused by temporal fluctuations. For example, when modeling housing prices in a city, $\beta_{j,t}$ would tolerate anomalous short-term fluctuations in prices, while large values of $\alpha_{jk,t}$ might be found near boundaries of disparate neighborhoods.

Adaptation to Streaming Data. In many applications, observed network snapshots arrive in a streaming fashion. Instead of recomputing all models on past snapshots whenever a new snapshot is observed, incorporating existing models to facilitate the new incoming snapshot is more efficient. Assume that we have learned models $\{\hat{x}_{j,t}\}_{j\in\mathcal{V}}$ for the τ-th snapshot, we then read observations of the next m snapshots indexed by $\tau+1,\cdots,\tau+m$ and attempt to learn models for them. To this intent, we deploy our ST-DANR formulation in Eq. (6.13) on a limited time interval $t=\tau,\cdots,\tau+m$, and then integrate established model at $t=\tau$ by fixing $\{x_{j,\tau}\}_{j\in\mathcal{V}}=\{\hat{x}_{j,\tau}\}_{j\in\mathcal{V}}$ and solving the remaining variables $\{x_{j,t}\}_{j\in\mathcal{V}}^{t=\tau+1,\cdots,\tau+m}$ in the corresponding problem. Note that in our setting, temporal messages are only transferable through consecutive snapshots, meaning that fixing the τ-th snapshot is the same as fixing all past snapshots.

Lastly, we point out that distributed ADMM-based solution for DANR (§2.2) can be adapted to ST-DANR, for either batch formulation (Eq. (6.13)) or streaming-case formulation, which readers can refer to in Appendix E.

Temporal Evolutionary Patterns. In this work, we consider the unsquared edge penalty term $\|x_{j,t}+\beta_{j,t}-x_{j,t+1}\|_2$ in $\mathcal{R}_T$ since it enforces temporal reconstruction, indicating sharp transitions or change points at particular timestamps. Similar to the spatial case, there could be multiple available alternatives to enforce different types of evolutionary patterns [208]. For example, replacing the unsquared norm with squared norm form $\|x_{j,t}+\beta_{j,t}-x_{j,t+1}\|_2^2$ will yield smoothly varying models along consecutive snapshots, and has been well studied in [209, 210].

6.1.6 Experiments

In this section, we present an experimental analysis of the proposed method (DANR) and evaluate its performance on various real-world tasks. We employ two housing price datasets from Bay Area and Sacrament Area to demonstrate an application that is particularly good fit for DANR. In addition to an improvement on the estimation task, DANR learns fine-grained neighborhood boundaries and reveals interesting insights into the network's heterogeneous structure. Lastly, in order to validate the efficacy of ST-DANR in the temporal setting, we conduct experiments with the geospatial and temporal database of Northeastern US lakes, from which we aim to estimate the water quality of lakes over 10 years. With these experiments, we validate the efficacy of the DANR method in the temporal setting, as well as the spatial setting.

Baseline Algorithms. In spatial network scenarios, we compare DANR against three baselines: network lasso (NL) [182], robust multi-task feature learning (rMTFL) [211], factorized multi-task learning (FORMULA) [212]. In spatial-temporal network scenarios, we compare ST-DANR with two widely-applied temporal regularizers in the literature, which are detailed later.

Parameters Setting. For both NL and DANR, we tune λ parameters from 10^{-3} to 10^{2} where $\lambda^{n+1} = 1.3\lambda^{n}$. Concerning the DANR, for each value of λ, we further tune μ parameters from 0.3 to 1, where $\mu^{n+1} = \mu^{n} + 0.02$. Lastly, we set $p = 3$. We follow the same strategy for both spatial (DANR) and spatio-temporal (ST-DANR) variants of the proposed method. For all penalty parameters in rMTFL and FORMULA, we tune them in the same way as λ in DANR. In addition, we vary the number of factorized base models in FORMULA from 1 to 50 to achieve its best performance. For each dataset, we standardize all features and response variables to zero mean and unit variance.

Spatial: Housing Rental Price Estimation

In this section, our goal is to jointly (i) estimate the rental prices of houses by leveraging their geological location and the set of features; (ii) discover boundaries between neighborhoods. The intuition is that the houses in the same neighborhood often tend to have similar pricing models. However, such neighborhoods can have complex underlying structures (as described later in this section), which imposes additional challenges in learning accurate models.

Dataset. We experiment with two largely populated areas in Northern California: the Greater Sacramento Area (SAC) and the Bay Area (BAY). The anonymized data is provided by the property management software company *Appfolio Inc*, from which we further sample houses with at least one signed rental agreement during the year 2017. The resulting dataset covers 3849 houses in SAC and 1498 houses in BAY. Concerning the houses that have more than one rental agreement signed during 2017, we average the rental prices listed in all the agreements. Each house holds the information about its location (latitude/longitude), number of bedrooms, number of bathrooms, square footage, and the rental price. We regard these areas (SAC and BAY) as two separate datasets for the remainder of this section. We also randomly split 20 % of the houses in each dataset for testing.

Network Construction. After excluding test houses from both datasets, we construct two networks (one for each dataset) based on the houses' locations. An undirected edge exists between node i and j, if at least one of them is in the set of 10 nearest neighbors of the other. (Note that node j being one of the 10 nearest neighbors of i doesn't necessarily imply that node i is also in the set of nearest neighbors of j.) In the resulting networks, the average degree of a node is 12.16 for SAC and 12.04 for BAY. Additionally, we construct two versions of these networks, *weighted* and *unweighted.* While the weighted network has edge weights inversely proportional to the distances between houses, the unweighted network ignores the proximity between houses, and thus weights on all the edges are 1.

Optimization Problem. The model at each house estimates its rental price by solving a linear regression problem. More specifically, at each node i we learn a 4-dimensional model $x_i = [a_i, b_i, c_i, d_i]$. x simply represents the coefficients of each feature, which later is used to estimate the rental price p_i as follows:

$$\begin{aligned}\overline{p_i} = a_i \cdot (\#bedrooms) + b_i \cdot (\#bathrooms) \\ + c_i \cdot (squarefootage) + d_i,\end{aligned}$$

where d_i is the bias term. The training objective is

$$\min_{x,\alpha} \sum_{i \in \mathcal{V}} \|\overline{p_i} - p_i\|_2^2 + c \cdot \|x_i\|_2^2 + \lambda \cdot \mathcal{R}_S(x, \alpha)$$

where $\mathcal{R}_S$ represents the proposed network regularization term (see Eq. 6.3), while c is the regularization weight to prevent over-fitting.

Test Results. Once training converges, we predict the rental prices on the test set. To do that, we connect each house in the test set to its 10 nearest neighbors in the training set. We then infer the new model x_j by taking the average of the models on its neighbors: $x_j = (1/|N(j)|) \sum_{k \in N(j)} x_k$. The model x_j is further used to estimate the rental price of the corresponding house. Alternatively, one can also infer x_j by solving $\min_{x_j} \sum_{k \in N(j)} \|x_j - x_k^*\|_2$, while keeping the models on neighbors fixed. However, we empirically find that simply averaging the neighbors' models performs better for both methods in this particular setting. We compute Mean Squared Error (MSE) over test nodes to evaluate the performance.

Table 6.1 displays the test results of eight different settings on both datasets. As shown, local & global estimations and rMTFL method produce high errors for both datasets. We further apply FORMULA and Network Lasso (NL) methods to both weighted and unweighted versions of the networks, while the proposed method is only applied to the unweighted version. Notice that the network weights are irrelevant for the local & global estimation settings and the rMTFL method. Intuitively, both FORMULA

Method	BAY	SAC
Local estimation ($\lambda = 0$)	0.5984	0.6250
Global estimation ($\lambda > \lambda_{critical}$)	0.4951	0.5403
rMTFL	0.4774	0.4115
FORMULA (unweighted)	0.4446	0.3503
FORMULA (weighted)	0.4181	0.3379
Network Lasso (unweighted)	0.4392	0.3273
Network Lasso (weighted)	0.4173	0.3022
DANR (unweighted)	**0.4106**	**0.2978**

Table 6.1: MSE for housing rental price prediction on test set.

and NL performs better on the weighted setting compared to the unweighted setting for both datasets. As shown, the rental price estimation errors achieved by the NL are 0.4173 (weighted) vs. 0.4392 (unweighted) for BAY and 0.3022 (weighted) vs. 0.3273 (unweighted) for SAC. These results suggest that the pre-defined weights on these networks help to learn more accurate models on houses.

However, we further argue that although such pre-defined weights improve the overall clustering performance, they don't account for more complex clustering scenarios, e.g., two nearby houses falling into different school districts or some houses having higher values compared to their neighbors due to geography, e.g., having a view of the city. That being said, DANR outperforms all the other baselines and achieves the smallest errors for both datasets; 0.4106 for BAY and 0.2978 for SAC. Especially, the DANR (unweighted) even outperforms the weighted version of the baseline approaches by a notable margin. This reveals that the DANR indeed accounts for such heterogeneities in data and provides enhanced clustering of the network.

Figure 6.2 shows examples of two complex scenarios that are captured by the DANR from the SAC network. We use a color code to represent the clusters in the network, where the same colored houses (i, j) are in consensus on their models ($x_i = x_j$). In the left subfigure, the house shown in yellow uses a different model than all of its neighbors.

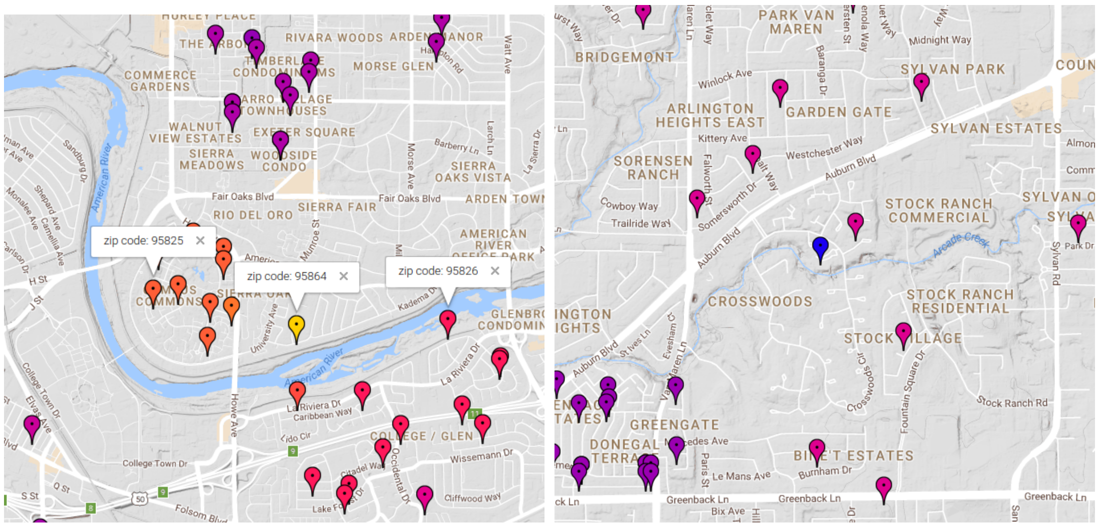


Figure 6.2: Examples of complex neighborhood structures captured by the DANR. In left, the house shown in yellow resides at the border of three different area codes. In right, the creek side house (colored in blue) differs from all of its neighbors, possibly due to its appealing location.

Interestingly, it resides at the border of three different area codes. The area code for this house is 95864, while the area code on its west is 95825 and 95826 on its south-east. Additionally, we observe similar heterogeneous behaviors in some houses that are near creeks, lakes, and rivers. As an example, Figure 6.2 (right) displays a creek side house (colored in blue), which again uses a different underlying model from its neighbors.

Spatio-Temporal: Water Quality Estimation of Northeastern US Lakes

We now evaluate our method in spatio-temporal setting, where the aim is to dynamically estimate the water quality of Northeastern US lakes over years. We follow the same procedure as before, but have an additional temporal regularization term in our objective that allows nodes to obtain signals from past snapshots of the network, along with their neighbors in the current snapshot.

Dataset and Network. We experiment with the geospatial and temporal dataset of lakes in 17 states in the Northeastern United States, called LAGOS [213]. The dataset holds extensive information about the physical characteristics, various nutrient concentration measurements (water quality), ecological context (surrounding land use/cover, climate etc.), and location of lakes; from which we select the following available set of features: {*max depth, surface area, elevation, annual mean temperature, % of agricultural land, % of urban land, % of forest land, % of wetland*}. The water quality metric to estimate is the summerly mean *chlorophyll concentration* of the lakes. We represent the feature vector with $w \in \mathbb{R}^8$ and the water quality score with $y \in \mathbb{R}$.

During our experiments, we consider a 10-year period between 2000 and 2010. Due to sparsity in the data, we allow 2 years range between the two consecutive snapshots of the network. This results in total of 1039 lakes with all the aforementioned features and the water quality measurements available for each of the selected years. After randomly splitting 20% of the lakes for testing, we build our network by using the latitude/longitude information of lakes, where each lake (node) is connected to 10 nearest lakes.

Optimization Problem. After constructing the network, we now aim to *dynamically* estimate the water quality ($y_{i,t}$) of lakes by using their feature vectors ($w_{i,t}$). More formally, for each year $t \in [2000, 2002, \cdots, 2010]$, we learn a model $x_{i,t} \in \mathbb{R}^8$ per node by solving the following spatio-temporal problem in a *streaming* fashion:

$$\min_{x,\alpha,\beta} \sum_{i \in \mathcal{V}} f(x_{i,t}, w_{i,t}, y_{i,t}) + \lambda_1 \cdot \mathcal{R}_S^t(x, \alpha) + \lambda_2 \cdot \mathcal{R}_T^t(x, \beta) \tag{6.16}$$

where f is the linear regression objective and $\mathcal{R}_S$ is the DANR term applied on the spatial edges. $\mathcal{R}_T$ is the temporal regularization term, for which we consider two formulations as described next.

Test results. For each year, we first solve the spatial problem (Eq. 6.16 without the temporal term $\mathcal{R}_T$) and report the Mean Squared Errors in Table 6.2. Note that the test nodes are again inferred by averaging the neighbors' models in the current snapshot. Later, in order to successfully assess the improvements gained by the temporal discrepancy-buffering variables (β), we solve the above spatio-temporal problem with three different temporal regularizers:

DANR+T-SON: DANR with temporal sum-of-norms regularizer where
$\mathcal{R}_T^t(x, \cdot) = \sum_{i \in \mathcal{V}} \|x_{i,t} - \hat{x}_{i,t-1}\|_2$

DANR+T-SOS: DANR with temporal sum-of-squares regularizer where
$\mathcal{R}_T^t(x, \cdot) = \sum_{i \in \mathcal{V}} \|x_{i,t} - \hat{x}_{i,t-1}\|_2^2$

ST-DANR: Spatio-temporal discrepancy-aware network regularizer where
$\mathcal{R}_T^t(x, \beta) = \mu_2 \cdot \sum_{i \in \mathcal{V}} \|x_{i,t} - \hat{x}_{i,t-1} + \beta_{i,t}\|_2 + (1 - \mu_2) \cdot \|\beta\|_{1,p}$

DANR+T-SON and DANR+T-SOS approaches simply apply two widely adopted temporal regularizers on the temporal edges (the sum-of-norms regularizer [197] and sum-of-squares reqularizer [210, 209] respectively), while ST-DANR includes the discrepancy-buffering variables on both spatial and temporal edges. Recall that we solve the Eq. 6.16 in a streaming fashion for simplicity, i.e., each snapshot of the network solves the problem while holding the models learned on the previous snapshot (if available) fixed. Potentially, the performance can further be improved by allowing updates on the previous snapshots.

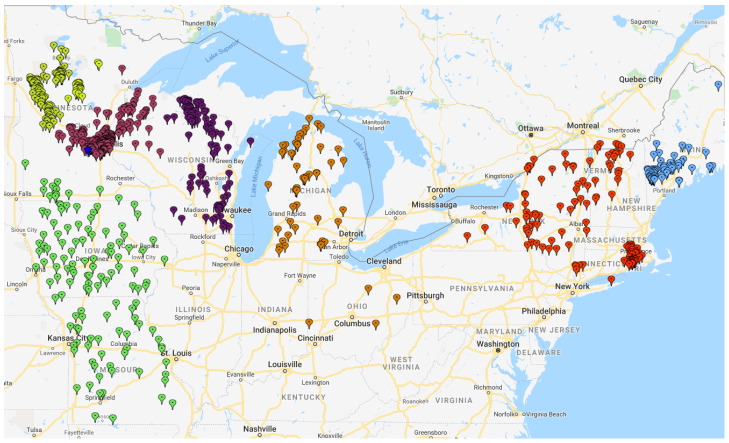
As we can see from the Table 6.2, leveraging the temporal connections between the two consecutive snapshots significantly improves the performance. The DANR+T-SON outperforms the DANR by 8%-14%, while DANR+TSOS outperforms the DANR by 10%-15%. Moreover, the ST-DANR consistently outperforms both baselines for all the years. This confirms that the proposed formulation accounts for the heterogeneous nature of the temporal networks where some group of nodes exhibits different evolution patterns than the others.

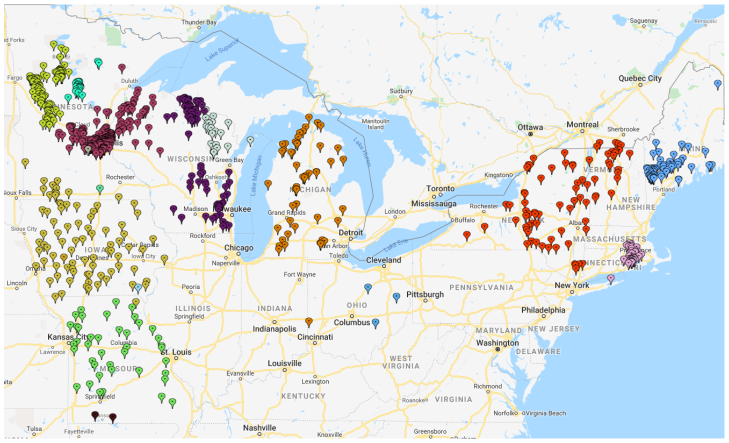
Figure 6.3 further displays the models (color-coded) learned on three consecutive snapshots, corresponding to years 2000, 2002 and 2004. In 2000, the formed clusters don't go much beyond the boundaries of the states, resulting in coarse clustering of the network.

Method	2000	2002	2004	2006	2008	2010
DANR	0.8479	0.9033	0.6384	0.6722	0.4556	0.4113
+ T-SON	N/A	0.8308	0.5744	0.6061	0.4109	0.3517
+ T-SOS	N/A	0.8045	0.5618	0.6045	0.3978	0.3476
ST-DANR	N/A	0.8037	0.5604	0.5866	0.3844	0.3311

Table 6.2: MSE for water quality estimation over years.

Yet some states such as Missouri and Iowa are grouped into one cluster (shown in green). This suggests that accurately estimating the chlorophyll concentration of lakes based on the selected set of features is indeed challenging. Specifically, the estimation problem depends on several other external factors that potentially affect the volume of plants and algae in lakes; cultural eutrophication, nutrient inputs from human activities to name a few [213]. However, as it can be seen from the models learned in 2002 and 2004, the clustering performance improves once we allow temporal regularization to synchronize models between two consecutive snapshots, which is analogous with the reduction in errors over years as reported in Table 6.2. Overall, we observe a split of clusters over time, e.g., in Wisconsin, Minnesota, Iowa and New England. Additionally, a dark brown cluster in south Missouri begins to appear in 2002 and further spreads north in 2004. This indicates that by leveraging the temporal connections, the ST-DANR learns improved models and clustering while allowing for heterogeneity in group level evolutions of nodes.


(a) Year 2000


(b) Year 2002

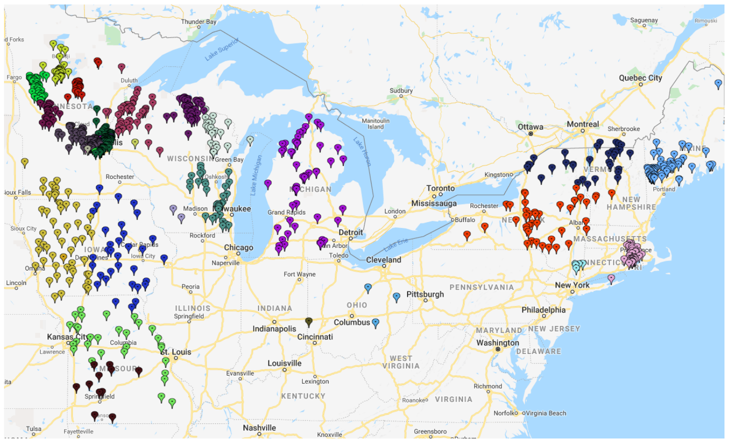

(c) Year 2004

Figure 6.3: Evolution of clustering captured by the ST-DANR over years.

6.1.7 Conclusion

We propose discrepancy-aware network regularization (DANR) approach for spatio-temporal networks. By introducing discrepancy-buffering variables, the DANR automatically compensates for inaccurate prior knowledge encoded in edge weights, and enables modeling heterogeneous temporal patterns of network clusters. We develop a scalable algorithm based on alternating direction method of multipliers (ADMM) to solve the proposed problems, which enjoys analytic solutions for decomposed subproblems and employs a distributed update scheme on nodes and edges. Our experimental results show that DANR yields enhanced models by providing robustness towards missing or inaccurate edge weights and successfully interprets structural changes in evolving networks.

6.2 Combining Physics and Machine Learning for Network Flow Estimation

6.2.1 Introduction

In many applications, ranging from road traffic to supply chains to power networks, the dynamics of flows on edges of a graph is governed by physical laws/models [165, 166]. For instance, the LWR model describes equilibrium equations for road traffic [214, 215]. However, it is often difficult to fully observe flows in these applications and, as a result, they rely on off-the-shelf machine learning models to make predictions about missing flows [216, 217]. A key limitation of these machine learning models is that they disregard the physics governing the flows. So, the question arises: can we combine physics and machine learning to make better flow predictions?

This paper investigates the problem of predicting missing edge flows based on partial observations and the underlying domain-specific physics defined by flow conservation and edge features [218]. Edge flows depend on the graph topology due to a flow conservation law—i.e. the total in-flow at every vertex is approximately its total out-flow. Moreover, the flow at an edge also depends on its features, which might regularize the space of possible flow distributions in the graph. Here, we propose a model that learns how to predict missing flows from data using bilevel optimization [219] and neural networks. More specifically, features are given as inputs to a neural network that produces edge flow regularizers. Weights of the network are then optimized via reverse-mode differentiation based on a flow estimation loss from multiple train-validation pairs.

Our work falls under a broader effort towards incorporating physics knowledge to machine learning, which is relevant for natural sciences and engineering applications where data availability is limited [220]. Conservation laws (of energy, mass, charge, etc.) are essential to our understanding of the physical world. The classical Noether's theorem shows that such laws arise from symmetries in nature [221]. However, flow estimation, which is an inverse problem [222, 223], is ill-posed under conservation alone. Regularization enables us to apply domain-knowledge in the solution of inverse problems.

We motivate our problem and evaluate its solutions using two application scenarios. The first is road traffic networks [224], where vertices represent locations, edges are road segments, flows are counts of vehicles that traverse a segment and features include numbers of lanes and speed limits. The second scenario is electric power networks [178], where vertices represent power buses, edges are power lines, flows are amounts of power transmitted and edge features include resistances and lengths of lines. Irrigation channels, gas pipelines, blood circulation, supply chains, air traffic, and telecommunication networks are other examples of flow graphs.

Our contributions can be summarized as follows: (1) We introduce a missing flow estimation problem with applications in a broad class of flow graphs; (2) we propose a model for flow estimation that is able to learn the physics of flows by combining reverse-mode differentiation and neural networks; (3) we show that our model outperforms the best baseline by up to 18%; and (4) we provide evidence that our model learns interpretable physical properties, such as the role played by resistance in a power transmission network and by the number of lanes in a road traffic network.

6.2.2 Flow Estimation Problem

We introduce the flow estimation problem, which consists of inferring missing flows in a network based on a flow conservation law and edge features.

Flow Graph. Let $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{X})$ be a flow graph with vertices $\mathcal{V}$ ($n = |\mathcal{V}|$), edges $\mathcal{E}$ ($m = |\mathcal{E}|$), and edge feature matrix $\mathcal{X} \in \mathbb{R}^{m \times d}$, where $\mathcal{X}[e]$ are the features of edge e. A flow vector $\mathbf{f} \in \mathbb{R}^m$ contains the (possibly noisy) flow f_e for each edge $e \in \mathcal{E}$. In case $\mathcal{G}$ is directed, $\mathbf{f} \in \mathbb{R}^m_+$, otherwise, a flow is negative if it goes against the arbitrary orientation of its edge. We assume that flows are induced by the graph, and thus, the total flow—in plus out—at each vertex is approximately conserved:

$$\sum_{(v_i,u)\in\mathcal{E}} f_{(v_i,u)} \approx \sum_{(u,v_o)\in\mathcal{E}} f_{(u,v_o)}, \forall u \in \mathcal{V} \tag{6.17}$$

For a road network, flow conservation implies that vehicles mostly remain on the road.

Flow Estimation Problem. Given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{X})$ with partial flow observations $\hat{\mathbf{f}} \in \mathbb{R}^{m'}$ for a subset $\mathcal{E}' \subseteq \mathcal{E}$ of edges ($\hat{f}_e$ is the flow for $e \in \mathcal{E}'$, $m' = |\mathcal{E}'| < m$), predict flows for edges in $\mathcal{E} \setminus \mathcal{E}'$.

In our road network example, partial vehicle counts $\hat{f}$ might be measured by sensors placed at a few segments, and the goal is to estimate counts at the remaining segments. One would expect flows not to be fully conserved in most applications due to the existence of inputs and outputs, such as parking lots and a power generators/consumers. In case these input and output values are known exactly, they can be easily incorporated to our problem as flow observations. Moreover, if they are known approximately, we can apply them as priors (as will be detailed in the next section). For the remaining of this paper, we assume that inputs and outputs are unknown and employ flow conservation as an approximation of the system. Thus, different from classical flow optimization problems, such as min-cost flow [225], we assume that flows are conserved approximately.

Notice that our problem is similar to the one studied in [218]. However, while their definition also assumes flow conservation, it does not take into account edge features. We claim that these features play important role in capturing the physics of flows. Our main contribution is a new model that is able to learn how to regularize flows based on edge features using neural networks.

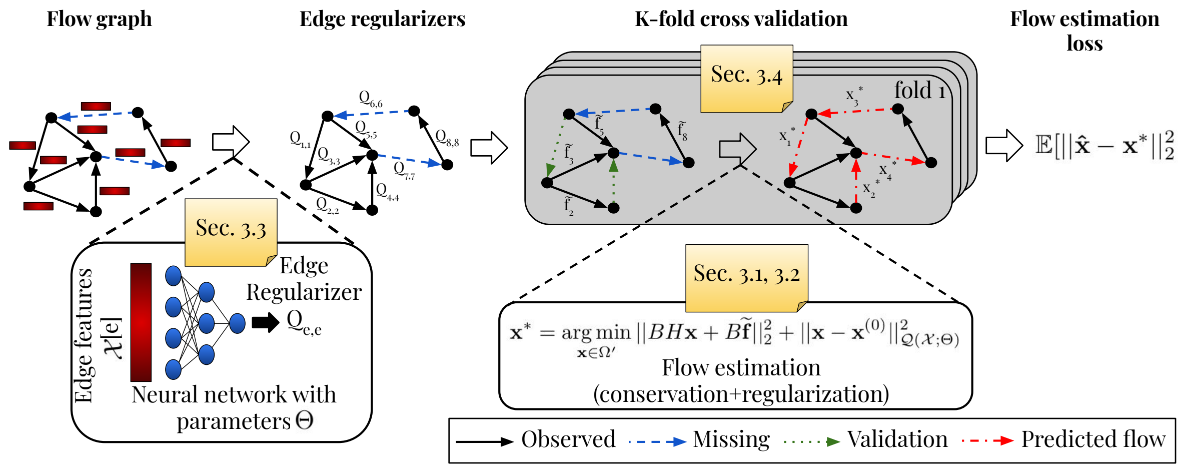

Flow graph
Edge regularizers
K-fold cross validation
Flow estimation loss
Sec. 3.4
fold 1
$\mathbb{E}[||\hat{\mathbf{x}} - \mathbf{x}^*||_2^2$
Sec. 3.3
Edge Regularizer
$Q_{e,e}$
Edge features $\mathcal{X}[e]$
Neural network with parameters Θ
Sec. 3.1, 3.2
$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\Omega'} ||BH\mathbf{x} + B\widetilde{\mathbf{f}}||_2^2 + ||\mathbf{x} - \mathbf{x}^{(0)}||^2_{\mathcal{Q}(\mathcal{X};\Theta)}$
Flow estimation (conservation+regularization)
Observed
Missing
Validation
Predicted flow

Figure 6.4: Summary of the proposed approach for predicting missing flows in a graph based on partial observations and edge features. We learn to combine features and a flow conservation law, which together define the physics of the flow graph. A regularization function $\mathcal{Q}(\mathcal{X};\Theta)$ modeled as a neural network with parameters Θ takes as input edge features $\mathcal{X}[e]$. A flow estimation algorithm applies the regularization, partial observations $(\widetilde{\mathbf{f}})$, prior flows $(\mathbf{x}^{(0)})$ and flow conservation to predict missing flows $\mathbf{x}$. Network parameters Θ are learned based on a K-fold cross validation loss with respect to validation flows $\hat{\mathbf{x}}$. Our model is trained end-to-end using reverse-mode differentiation.

6.2.3 Our Approach: Physics+Learning

In this section, we introduce our approach for the flow estimation problem, which is summarized in Figure 6.4. We formulate flow estimation as an optimization problem (Section 6.2.3.1), where the interplay between the flow network topology and edge features is defined by the physics of flow graphs. Flow estimation is shown to be equivalent to a regularized least-squares problem (Section 6.2.3.2). Moreover, we describe how the effect of edge features and the graph topology can be learned from data using bilevel optimization and neural networks in Section 6.2.3.3. Finally, we propose a reverse-mode differentiation algorithm for flow estimation in Section 6.2.3.4.

6.2.3.1 Flow Estimation via Optimization

The extent to which flow conservation holds for flows in a graph is known as *divergence* and can be measured using the *oriented incidence matrix* $B \in \mathbb{R}^{n\times m}$ of $\mathcal{G}$. The matrix is defined as follows, $B_{ij} = 1$ if $\exists u$ such that $e_j = (v_i, u) \in \mathcal{E}$, $B_{ij} = -1$ if $\exists u$ such that $e_j = (u, v_i) \in \mathcal{E}$, and $B_{ij} = 0$, otherwise. Given B and $\mathbf{f}$, the divergence at a vertex u can be computed as:

$$(B\mathbf{f})_u = \sum_{(v_i,u)\in\mathcal{E}} f_{(v_i,u)} - \sum_{(u,v_o)\in\mathcal{E}} f_{(u,v_o)} \tag{6.18}$$

And thus, we can compute the total (squared) divergence in the graph as $||B\mathbf{f}||_2^2 = \mathbf{f}^\intercal B^\intercal B\mathbf{f} = \sum_{u\in\mathcal{V}}((B\mathbf{f})_u)^2$. One could try to solve the flow estimation problem by minimizing $||B\mathbf{f}||_2^2$ while keeping the observed flows fixed, however, this problem is ill-posed—there might be multiple solutions to the optimization. The standard approach in such a scenario is to resort to regularization. In particular, we apply a generic regularization function Φ with parameters Θ as follows:

$$\mathbf{f}^* = \operatorname*{argmin}_{\mathbf{f}\in\Omega} ||B\mathbf{f}||_2^2 + \Phi(\mathbf{f}, \mathcal{X}; \mathbf{f}^{(0)}; \Theta) \quad st. \quad f_e = \hat{f}_e, \forall e \in \mathcal{E}' \tag{6.19}$$

where Ω is the domain of $\mathbf{f}$, $\mathbf{f}^{(0)} \in \mathbb{R}^m$ is a prior for flows, f_e ($\hat{f}_e$) are entries of $\mathbf{f}$ ($\hat{\mathbf{f}}$) for edge e and the constraint guarantees that observed flows are not changed. Priors $\mathbf{f}^{(0)}$, not be confused with observed flows $\hat{\mathbf{f}}$, should be set according to the application (e.g., as zero, based on a black-box model or historical data). Regarding the domain Ω, we consider $\Omega = \mathbb{R}^m$ and $\Omega = \mathbb{R}^m_+$. The second case is relevant for directed graphs—when flows must follow edge orientations (e.g., traffic).

In [218], the authors set $\Phi(\mathbf{f}, X, \mathbf{f}^{(0)}; \Theta)$ as $\lambda^2||\mathbf{f}||_2^2$ for a regularization parameter λ, which implies a uniform zero prior with an L_2 penalty over edges. We claim that the regularization function plays an important role in capturing the physics of flow graphs.

As an example, for a power network, Φ should account for the resistance of the lines. Thus, we propose learning the regularization from data. Our approach is based on a least-squares formulation, which will be described next.

6.2.3.2 Regularized Least-squares Formulation

Flow estimation problem can be viewed as an *inverse problem* [222]. Let $\mathbf{x} \in \mathbb{R}^{m-m'}$ be the vector of missing flows and $H \in \mathbb{R}^{m \times m-m'}$ be a matrix such that $H_{ij} = 1$ if f_i maps to x_j (i.e., they are associated to the same edge), and $H_{i,j} = 0$, otherwise. Moreover, let $\widetilde{\mathbf{f}} \in \mathbb{R}^m$ be such that $\widetilde{f}_e = \hat{f}_e$ if $e \in \mathcal{E}'$ and $\widetilde{f}_e = 0$, otherwise. Using this notation, we define flow estimation as $BH\mathbf{x} = -B\widetilde{\mathbf{f}} + \epsilon$, where BH is a forward operator, projecting $\mathbf{x}$ to a vector of vertex divergences, and $-B\widetilde{\mathbf{f}} + \epsilon$ is the observed data, capturing (negative) vertex divergences for observed flows. The error ϵ can be interpreted as noise in observations or some level of model misspecification.

We can also define a regularized least-squares problem with the goal of recovering missing flows $\mathbf{x}$:

$$\mathbf{x}^* = \underset{\mathbf{x} \in \Omega'}{\operatorname{argmin}} \, ||BH\mathbf{x} + B\widetilde{\mathbf{f}}||_2^2 + ||\mathbf{x} - \mathbf{x}^{(0)}||^2_{\mathcal{Q}(\mathcal{X};\Theta)} \tag{6.20}$$

where Ω' is a projection of the domain of $\mathbf{f}$ to the space of $\mathbf{x}$, $||\mathbf{x}||_M^2 = \mathbf{x}^\intercal M \mathbf{x}$ is the matrix-scaled norm of $\mathbf{x}$ and $\mathbf{x}^{(0)} \in \mathbb{R}^{m-m'}$ are priors for missing flows. The regularization function $\Phi(\mathbf{f}, \mathcal{X}; \mathbf{f}^{(0)}, \Theta)$ has the form $||\mathbf{x} - \mathbf{x}^{(0)}||^2_{\mathcal{Q}(\mathcal{X};\Theta)}$, where the matrix $\mathcal{Q}(\mathcal{X};\Theta)$ is a function of parameters Θ and edge features $\mathcal{X}$. We focus on the case where $\mathcal{Q}(\mathcal{X};\Theta)$ is non-negative and diagonal.

Equation 6.20 has a Bayesian interpretation, with $\mathbf{x}$ being a maximum likelihood estimate under a Gaussian assumption—i.e., $\mathbf{x} \sim N(\mathbf{x}^{(0)}, \mathcal{Q}(\mathcal{X};\Theta)^{-1})$ and $B\widetilde{\mathbf{f}} \sim N(0, I)$ [222]. Thus, $\mathcal{Q}(\mathcal{X};\Theta)$ captures the variance in flow observations $\hat{\mathbf{f}}$ in prior estimates $\mathbf{f}^{(0)}$

compared to the one. This allows the regularization function to adapt to different edges based on their features. For instance, in our road network example, $Q(\mathcal{X};\Theta)$ might place a lower weight on flow conservation for flows at a road segment with a small number of lanes, which are possible traffic bottlenecks.

Given the least-squares formulation described in this section, how do we model the regularization function $\mathcal{Q}$ and learn its parameters Θ? We would like $\mathcal{Q}$ to be expressive enough to be able to capture complex physical properties of flows, while Θ to be computed accurately and efficiently. We will address these challenges in the remaining of this paper.

6.2.3.3 Bilevel Optimization for Meta-learning the Physics of Flows

This section introduces a model for flow estimation that is able to learn the regularization function $\mathcal{Q}(\mathcal{X};\Theta)$ in Equation 6.20 from data using bilevel optimization and neural networks.

Bilevel formulation. We learn the parameters Θ that determine the regularization function $\mathcal{Q}(\mathcal{X};\Theta)$ using the following bilevel optimization formulation:

$$\Theta^* = \underset{\Theta}{\operatorname{argmin}}\, \mathbb{E}[||\hat{\mathbf{x}} - \mathbf{x}^*||_2^2] \tag{6.21}$$

$$\text{st.} \quad \mathbf{x}^* = \underset{\mathbf{x}\in\Omega'}{\operatorname{argmin}}\, ||BH\mathbf{x} + B\widetilde{\mathbf{f}}||_2^2 + ||\mathbf{x} - \mathbf{x}_0||^2_{Q(\mathcal{X};\Theta)} \tag{6.22}$$

where the inner (lower) problem is the same as Equation 6.20 and the outer (upper) problem is the expected loss with respect to ground truth flows $\hat{\mathbf{x}}$—which we estimate using cross-validation.

Notice that optimal values for parameters Θ and missing flows $\mathbf{x}$ are both unknown in the bilevel optimization problem. The expectation in Equation 6.21 is a function of multiple instances of the inner problem (Equation 6.22). Each inner problem instance has an optimal solution $\mathbf{x}^*$ that depends on parameters Θ. In general, bilevel optimization is not only non-convex but also NP-hard [226]. However, recent gradient-based solutions for bilevel optimization have been successfully applied to large-scale problems, such as hyper-parameter optimization and meta-learning [227, 228]. We will first describe how we model the function $\mathcal{Q}(\mathcal{X};\Theta)$ and then discuss how this problem can be solved efficiently using reverse-mode differentiation.

We propose to model $\mathcal{Q}(\mathcal{X};\Theta)$ using a neural network, where $\mathcal{X}$ are inputs, Θ are learnable weights and the outputs are diagonal entries of the regularization matrix. This is a natural choice due to the expressive power of neural nets [229, 230]. **Multi-Layer Perceptron (MLP).** An MLP-based $\mathcal{Q}(\mathcal{X};\Theta)$ has the following form:

$$\mathcal{Q}(\mathcal{X};\Theta) = diag(MLP(\mathcal{X};\Theta)) \tag{6.23}$$

where $MLP(\mathcal{X};\Theta) \in \mathbb{R}^{m-m'}$. For instance, $\mathcal{Q}(\mathcal{X};\Theta)$ can be a 2-layer MLP:

$$\mathcal{Q}(\mathcal{X};\Theta) = diag(a(b(\mathcal{X}W^{(1)})W^{(2)})) \tag{6.24}$$

where $\Theta = \{W^{(1)}, W^{(2)}\}$, $W^{(1)} \in \mathbb{R}^{d\times h}$, $W^{(2)} \in \mathbb{R}^{h\times 1}$, h is the number of nodes in the hidden layer, both a and b are activation functions, and the bias was omitted for convenience.

Graph Neural Network (GNN). The MLP-based approach assumes that each entry $[\mathcal{Q}(\mathcal{X};\Theta)]_{e,e}$ associated to an edge e is a function of its features $\mathcal{X}[e]$ only. However, we are also interested in how entries $[\mathcal{Q}(\mathcal{X};\Theta)]_{e,e}$ might depend on the features of neighborhood of e in the flow graph topology. Thus, we consider the case where $\mathcal{Q}(\mathcal{X};\Theta)$ is a GNN. For instance, we apply a 2-layer spectral Graph Convolutional Network (GCN) with Chebyshev convolutions [231, 159, 232]:

$$\mathcal{Q}(\mathcal{X};\Theta) = diag\left(ReLU\left(\sum_{z'=1}^{Z'} T_{z'}(\widetilde{L})ReLU\left(\sum_{z=1}^{Z} T_z(\widetilde{L})\mathcal{X}W_z^{(1)}\right)W_{z'}^{(2)}\right)\right) \tag{6.25}$$

where $\widetilde{L} = 2/\lambda_{max}L - I$, L is the normalized Laplacian of the undirected version of the line graph $\mathcal{G}'$ of $\mathcal{G}$, λ_{max} is the largest eigenvalue of L, $T_z(\widetilde{L})$ is a Chebyshev polynomial of $\widetilde{L}$ with order z and $W_z^{(i)}$ is the matrix of learnable weights for the z-th order polynomial at the layer i. In a line graph, each vertex represents an edge of the undirected version of $\mathcal{G}$ and two vertices are connected if their corresponding edges in $\mathcal{G}$ are adjacent. Morever $L = I - D^{-1/2}AD^{-1/2}$, where A and D are the adjacency and degree matrices of $\mathcal{G}'$. Chebyshev polynomials are defined recursively, with $T_z(y) = 2yT_{z-1}(y) - T_{z-2}(y)$ and $T_1(y) = y$.

In our experiments, we have also applied the more popular non-spectral graph convolutional operator [159] but preliminary results have shown that the Chebyshev operator achieves better performance in flow estimation.

6.2.3.4 Flow Estimation Algorithm

We now focus on how to solve our bilevel optimization problem (Equations 6.21 and 6.22). Our solution applies gradient-based approaches (e.g., SGD [233], Adam [66]) and, for simplicity, our description will be based on the particular case of Gradient Descent and assume a zero prior ($\mathbf{x}^{(0)} = \mathbf{0}$). A key challenge in our problem is to efficiently approximate the gradient of the outer objective with respect to the parameters Θ, which, by the chain rule, depends on the gradient of the inner objective with respect to Θ.

We first introduce extra notation to describe the outer problem (Equation 6.21). Let $(\hat{\mathbf{f}}_k, \hat{\mathbf{g}_k})$ be one of K train-validation folds, both containing ground-truth flow values, such that $\hat{\mathbf{f}}_k \in \mathbb{R}^p$ and $\hat{\mathbf{g}_k} \in \mathbb{R}^q$. For each fold k, we apply the inner problem (Equation 6.22) to estimate missing flows $\mathbf{x}_k$. Estimates for all folds are concatenated into a single vector $\mathbf{x} = [\mathbf{x}_1; \mathbf{x}_2; \dots; \mathbf{x}_K]$ and the same for validation sets $\hat{\mathbf{g}} = [\hat{\mathbf{g}_1}; \hat{\mathbf{g}_2}; \dots \hat{\mathbf{g}_K}]$. We define a matrix $R \in \mathbb{R}^{q\times(m-m')}$ such that $R_{ij} = 1$ if prediction x_j corresponds to validation flow $\hat{g}_i$ and $R_{ij} = 0$, otherwise. Using this representation, we can approximate the expectation in the outer objective as $\Psi(\mathbf{x}, \Theta) = (1/K)||R\mathbf{x} - \hat{\mathbf{g}}||_2^2$, where $\mathbf{x}$ depends implicitly on Θ. We also introduce $\Upsilon_\Theta(\mathbf{x})$ as the inner problem objective. Moreover, let $\Gamma_j(\mathbf{x}_{k,j-1}, \Theta_{i-1})$ be one step of gradient descent for the value of $\mathbf{x}_k$ at iteration j with learning rate β:

$$\begin{aligned}\Gamma_j(\mathbf{x}_{k,j-1}, \Theta_{i-1}) &= \mathbf{x}_{k,j-1} - \beta\nabla_{\mathbf{x}}\Upsilon_\Theta(\mathbf{x}_{k,j}) \\ &= \mathbf{x}_{k,j-1} - 2\beta[H_k^\intercal B^\intercal(BH_k\mathbf{x}_{k,j-1} + B\widetilde{\mathbf{f}}_k) + 2Q_k\mathbf{x}_{k,j-1}]\end{aligned}$$

where H_k, Q_k and $\widetilde{\mathbf{f}}_k$ are the matrix H, a sub-matrix of $\mathcal{Q}(\mathcal{X}; \Theta_{i-1})$ and the observed flows vector $\widetilde{\mathbf{f}}$ (see Section 6.2.3.2) for the specific fold k. We have assumed the domain (Ω') of flows $\mathbf{x}_{k,j}$ to be the set of real vectors. For non-negative flows, we add the appropriate proximal operator to Γ_j.

Our algorithm applies *Reverse-Mode Differentiation* (RMD) [234, 219] to estimate $\nabla_\Theta\Psi$ and optimizes Θ also using an iterative algorithm. The main idea of RMD is to first unroll and store a finite number of iterations for the inner problem $\mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_J$ and then reverse over those iterations to estimate $\nabla_\Theta\Psi$, which is computed as follows:

$$\nabla_{\mathbf{x}_J,\Theta}\Psi(\mathbf{x}_J, \Theta_i) = \nabla_{\mathbf{x}}\Psi(\mathbf{x}_J, \Theta_i)\sum_{j=1}^{J}\left(\prod_{s=j+1}^{J}\frac{\partial\Gamma_s(\mathbf{x}_{s-1}, \Theta_i)}{\partial\mathbf{x_{s-1}}}\right)\frac{\partial\Gamma_j(\mathbf{x}_{j-1}, \Theta_i)}{\partial\Theta}$$

In particular, our reverse iteration is based on the following equations:

$$\begin{aligned}
\nabla_{\mathbf{x}}\Psi(\mathbf{x}_J, \Theta_i) &= (2/K)R^{\intercal}(R\mathbf{x}_J - \hat{\mathbf{g}}) \\
\frac{\partial\Gamma_s(\mathbf{x}_{s-1}, \Theta_i)}{\partial\mathbf{x}_{\mathbf{s-1}}} &= I - 2\beta(H^{\intercal}B^{\intercal}BH + 2\mathcal{Q}(\mathcal{X};\Theta_i)) \\
\frac{\partial\Gamma_j(\mathbf{x}_{j-1}, \Theta_i)}{\partial\Theta} &= -4\beta(\partial\mathcal{Q}(\mathcal{X};\Theta_i)/\partial\Theta)\mathbf{x}_{j-1}
\end{aligned}$$

where $\partial\mathcal{Q}(\mathcal{X};\Theta_i)/\partial\Theta$ is the gradient of the regularization function $\mathcal{Q}(\mathcal{X};\Theta)$ evaluated at Θ_i. In our case, this gradient is the same as the neural network gradients and is omitted here for convenience.

Algorithm 5 RMD Algorithm for Flow Estimation

```
Require: Flow network $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{X})$, train-validation folds $\{(\hat{\mathbf{f}}_k, \hat{\mathbf{g}}_k)\}_{k=1}^{K}$, number of outer
    iterations $T$ and inner iterations $J$, learning rates $\alpha$ and $\beta$
Ensure: Regularization parameters $\Theta$
 1: Initialize parameters $\Theta_0$
 2: $\hat{\mathbf{g}} \leftarrow [\hat{\mathbf{g}_1}; \dots \hat{\mathbf{g}_K}]$
 3: $B \leftarrow$ incidence matrix of $\mathcal{G}$
 4: for outer iterations $i = 1, \dots T$ do
 5:     Initialize missing flows $\mathbf{x}_{k,0}$ for all $k$
 6:     for inner iterations $j = 1, \dots J$ do
 7:         for folds $k = 1, \dots K$ do
 8:             $\mathbf{x}_{k,j} \leftarrow \mathbf{x}_{k,j-1} - 2\beta[H_k^{\intercal}B^{\intercal}(BH_k\mathbf{x}_{k,j-1} + B\widetilde{\mathbf{f}}_k) + 2Q_k\mathbf{x}_{k,j-1}]$
 9:         end for
10:         $\mathbf{x}_j \leftarrow [\mathbf{x}_{1,j}; \dots \mathbf{x}_{K,j}]$
11:     end for
12:     $\mathbf{z}_J \leftarrow (2/K)R^T(R\mathbf{x}_J - \hat{\mathbf{g}})$
13:     for reverse inner iterations $j = J-1, \dots 1$ do
14:         $\overleftarrow{\Theta} \leftarrow \overleftarrow{\Theta} - 4\beta\mathbf{z}_{j+1}(\partial\mathcal{Q}(\mathcal{X};\Theta_{i-1})/\partial\Theta)\mathbf{x}_{j+1}$
15:         $\mathbf{z}_j \leftarrow \mathbf{z}_{j+1}[I - 2\beta(H^{\intercal}B^{\intercal}BH + \mathcal{Q}(\mathcal{X};\Theta_{i-1}))]$
16:     end for
17:     Update $\Theta_i \leftarrow \Theta_{i-1} - \alpha\overleftarrow{\Theta}$
18: end for
19: return parameters $\Theta_I$ =0
```

Algorithm 5 describes our RMD approach for flow estimation. It receives as inputs the flow network $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{X})$, K train-validation folds $\{(\hat{\mathbf{f}}_k, \hat{\mathbf{g}}_k)\}_{k=1}^{K}$, and also hyperparameters T, J, α, and β, corresponding to the number of outer and inner iterations, and learning rates for the outer and inner problem, respectively. Its output is a vector of optimal parameters Θ for the regularization function $\mathcal{Q}(\mathcal{X}; \Theta)$ according to the bilevel objective in Equations 6.21 and 6.22. We use $\overleftarrow{\Theta}$ to indicate our estimate of $\nabla_{\Theta}\Psi(\Theta_i)$. Iterations of the inner problem are stored for each train-validation fold in lines 4-12. Reverse steps, which produce an estimate $\overleftarrow{\Theta}$, are performed in lines 13-17. We then use $\overleftarrow{\Theta}$ to update our estimate of Θ in line 17. The time and space complexities of the algorithm are $O(TJKm)$ and $O(Jm)$, respectively, due to the cost of computing and storing the inner problem iterations.

As discussed in the previous section, bilevel optimization is non-convex and thus we cannot guarantee that Algorithm 5 will return a global optima. In particular, the learning objective of our regularization function $\mathcal{Q}(\mathcal{X}; \Theta)$ is non-convex—it is a neural network. However, the inner problem (Equation 6.22) in our formulation has a convex objective (least-squares). In [227], the authors have shown that this property implies convergence. We also find that our algorithm often converges to a good estimate of the parameters in our experiments.

6.2.4 Experiments

We evaluate our approaches for the flow estimation problem using two real datasets and a representative set of baselines and metrics.

6.2.4.1 Datasets

This section summarizes the datasets used in our evaluation. We normalize flow values to $[0, 1]$ and map discrete features to real vector dimensions using one-hot encoding.

Traffic: Vertices represent locations and directed edges represent road segments between two locations in Los Angeles County, CA.[1] Flows are daily average vehicle counts measured by sensors placed along highways in the year 2018. We assign each sensor to an edge in the graph based on proximity and other sensor attributes. Our road network covers the Los Angeles County area, with $5,749$ vertices, $7,498$ edges, of which $2,879$ edges (38%) have sensors. The following features were mapped to an 18-dimensional vector: lat-long coordinates, number of lanes, max-speed, and highway type (motorway, motorway link, trunk, etc.), in-degree, out-degree, and centrality (PageRank). The in-degree and centrality of an edge are computed based on its source vertex. Similarly, the out-degree of an edge is the out-degree of its target vertex.

Power: Vertices represent buses in Europe, undirected edges are power transmission lines and edge flows measure the total active power (in MW) being transmitted through the lines. The dataset is obtained from PyPSA-Eur [172, 235]—an optimization model of the European power transmission system—which generates realistic power flows based on solutions of optimal linear power flow problems with historical production and consumption data. Default values were applied for the PyPSA-Eur settings. The resulting graph has 2,048 vertices, 2,729 edges, and 14-dimensional feature vectors capturing resistance, reactance, length, and number of parallel lines, nominal power, edge degree etc.

6.2.4.2 Experimental Settings

Evaluation metrics: We apply Pearson's correlation (CORR), Mean Absolute Percentage Error (MAPE), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE) to compare ground-truth and predicted flows.

Baselines: Divergence minimization (*Div*) [218] maximizes flow conservation using a single regularization parameter λ, which we optimize using line search in a validation set

[1]Source: `https://www.openstreetmap.org`

of flows. Multi-Layer Perceptron (*MLP*) is a 2-layer neural network with ReLU activations for all layers that learns to predict flows based on edge features. Graph Convolutional Network (*GCN*) is a 2-layer graph neural network, also with ReLU activations and Chebyshev convolutions of degree 2, that learns to predict the flows using both edge features and the topology but disregarding flow conservation [159, 231]. We also consider two hybrid baselines. *MLP-Div* applies the predictions from *MLP* as priors to *Div*. Similarly, predictions from *GCN* are used as priors for *GCN-Div*. For both hybrid models, we also optimize the parameter λ.

Our approaches: We consider three variations of Algorithm 5. However, one important modification is that we perform the reverse iterations for each fold—i.e., folds are treated as batches in SGD. Bil-MLP and Bil-GCN apply our reverse-mode differentiation approach using an MLP and a GCN as a regularizer. Moreover, both approaches use zero as the prior $\mathbf{x}^{(0)}$. Bil-GCN-Prior applies the GCN predictions as flow priors. Architectures of the neural nets are the same as the baselines.

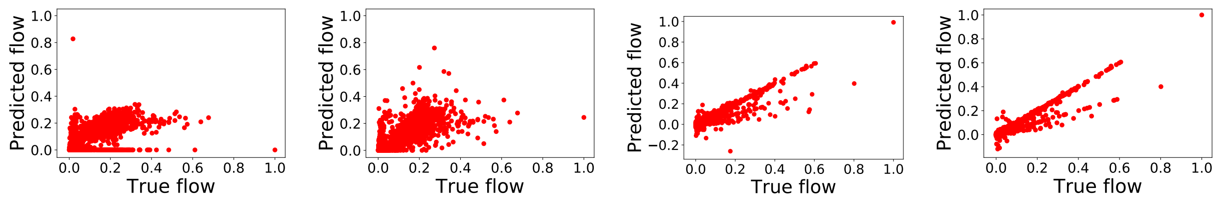
6.2.4.3 Flow Estimation Accuracy

Table 6.3 compares our methods and the baselines using several metrics over the Traffic and Power datasets. Values of CORR achieved by MLP and GCN for Traffic are missing because they were undefined—they generated predictions with zero variance for at least one of the train-test folds. All methods suffer from high MAPE errors for Power, which is due to an over-estimation of small flows. Bil-GCN achieves the best results in both datasets in terms of all metrics, with 6% and 18% lower RMSE than the best baseline for Traffic and Power, respectively. However, notice that Bil-MLP and Bil-GCN achieve very similar performance for Power and Bil-GCN-Prior does not outperform our other methods. We also show scatter plots with the true vs. predicted flows for the best approaches in Figure 6.5. Traffic has shown to be the more challenging dataset, which

	Traffic				Power			
Method	**RMSE**	**MAE**	**MAPE**	**CORR**	**RMSE**	**MAE**	**MAPE**	**CORR**
Div	0.071	0.041	1.23	0.76	0.034	0.015	1419.2	0.93
MLP	0.083	0.055	1.13	-	0.069	0.043	8334.5	0.61
GCN	0.066	0.040	0.94	-	0.064	0.043	5622.3	0.64
MLP-Div	0.066	0.041	1.51	0.81	0.033	0.015	1593.5	0.93
GCN-Div	0.071	0.048	1.69	0.81	0.033	0.015	1795.2	0.93
Bil-MLP	0.069	0.038	1.05	0.79	0.027	0.011	758.0	0.95
Bil-GCN	0.062	0.034	0.86	0.82	0.027	0.011	788.5	0.95
Bil-GCN-Prior	0.062	0.035	0.91	0.82	0.027	0.011	691.5	0.95

Table 6.3: Average flow estimation accuracy for the baselines (Div, MLP and GCN) and our methods (Bil-MLP, Bil-GCN and Bil-GCN-Prior) using the Traffic and Power datasets. RMSE, MAE and MAPE are errors (the lower the better) and CORR is a correlation (the higher the better). Values of correlation for MLP and GCN using Traffic were undefined. Bil-GCN (ours) outperforms the best baseline for all the metrics, with up to 18% lower RMSE than Div using Power.

can be explained in part by training data sparsity—only 38% of edges are labeled.

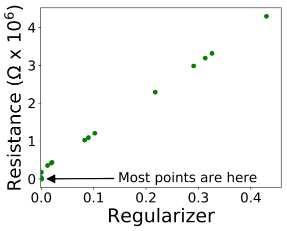


(a) GCN, Traffic (b) Bil-GCN, Traffic (c) Div, Power (d) Bil-GCN, Power

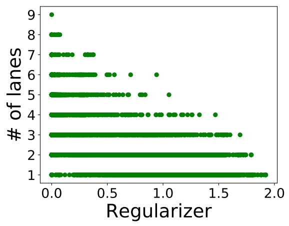
Figure 6.5: Scatter plots with true (x) and predicted (y) flows for two approaches on each dataset. The results are consistent with Table 6.3 and show that our methods are more accurate than the baselines.

6.2.4.4 Analysis of Regularizers

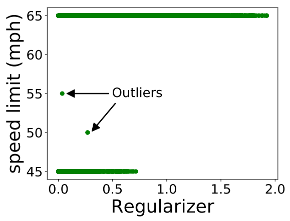
Figure 6.6 illustrates the regularization function learned by Bil-MLP. We focus on Bil-MLP because it can be analyzed independently of the topology. Figures 6.6a-6.6c show scatter plots where the x and y axes represent the value of the regularizer and features, respectively. For Power, Bil-MLP captures the effect of resistance over flows (Fig. 6.6a). However, only high values of resistance are mostly affected—that is the reason few points can be seen and also explains the good results for Div. We did not find a significant correlation for other features, with the exception of reactance, which is related to resistance. For Traffic, the model learns how the number of lanes constrains the flow at a road segment (Fig. 6.6b). Results for speed limit are more surprising, 45mph roads are less regularized (Fig. 6.6c). This is evidence that regularization is affecting mostly traffic bottlenecks in highways—with few lanes but a 65mph speed limit. To further investigate this result, we also show the regularizers over the Traffic topology in Figure 6.6d. High regularization overlaps with well-known congested areas in Los Angeles, CA (e.g., Highway 5, Southeast). These results are strong evidence that our methods are able to learn the physics of flows in road traffic and power networks.



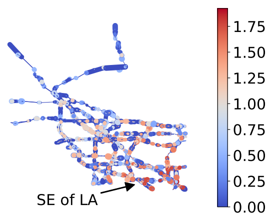
(a) Resistance, Power



(b) Lanes, Traffic



(c) Speed limit, Traffic



(d) Visualization, Traffic

Figure 6.6: Edge regularizer learned by Bil-MLP vs. features values (a-c) and visualization of regularizers on the Traffic topology (d). Our model is able to learn the effect of the resistance for Power. In Traffic, a higher number of lanes is correlated to less regularization and lower speed roads (45mph) are less regularized. The regularization is also correlated with congested areas in Los Angeles, CA.

6.2.5 Related Work

Flow graphs are quite ubiquitous in engineering, biomedical and social sciences. Two important properties of flow graphs are that their state space is defined by a graph topology and their dynamics are governed by the physics (or logic) of the problem of interest. We refer to [165] for a unified characterization of the mathematical treatment of flow graphs. Notice that these studies do not address the flow inference problem and their applications to real data is limited [236, 237]. Moreover, we focus on long term flows (e.g. daily vehicle traffic flows) and not on the dynamics. This simplifies the equations of our model to the conservation law.

Flow inference via divergence minimization was originally proposed in [218]. However, their work has not considered edge features and instead applied a single regularization parameter to the norm of the flow vector $\mathbf{f}$ in Equation 6.19. Our work leverages relevant edge features to learn the interplay between flow conservation and local predictions (priors). Thus, we generalize the formulation from [218] to the case of a learnable regularization function $\mathcal{Q}(\Theta, X)$. Our experiments show that the proposed approach achieves superior results in two datasets.

Flow optimization problems, such as min-cost flow, max-flow and multi-commodity flow, have a long history in computer science [225, 238]. These problems impose flow conservation as a hard constraint, requiring full knowledge of source and sink vertices and noiseless flow observations. Our approach relaxes these requirements by minimizing the flow divergence (see Equation 6.19). Moreover, our problem does not assume edge capacities and costs.

The relationship between flow estimation and inverse problems is of particular interest due to the role played by regularization [239] in the solution of ill-posed problems. Recent work on inverse problems has also focused on learning to regularize based on data and even learning the forward operator as well—see [223] for a review. The use of the expression "learning the physics" is also popular in the context of the universal differential equation framework, which enables the incorporation of domain-knowledge from scientific models to machine learning [240, 241, 220].

Bilevel optimization in machine learning has been popularized due its applications in hyperparameter optimization [242, 243]. In the last decade, deep learning has motivated novel approaches able to optimize millions of hyperparameters using gradient-based schemes [244, 228, 245]. Our flow estimation algorithm is based on reverse-mode differentiation, which is a scalable approach for bilevel optimization [219, 234, 244]. Another application of bilevel optimization quite related to ours is meta-learning [227, 246].

Our problem is also related to semi-supervised learning on graphs [247, 193, 248], which is the inference of vertex labels given partial observations. These approaches can be applied for flow estimation via the line graph transformation [218]. The duality between a recent approach for predicting vertex labels [182] and min-cost flows was shown in [249]. However, the same relation does not hold for flow estimation.

Graph neural network models, which generalize deep learning to graph data, have been shown to outperform traditional semi-supervised learning methods in many tasks [159, 250, 251]. These models have also been applied for traffic forecasting [216, 217, 252]. Different from our approach, traditional GNNs do not conserve flows. We show that our models outperform GNNs at flow prediction. Moreover, we also apply GNNs as a regularization function in our model.

6.2.6 Conclusion

We have introduced an approach for flow estimation on graphs by combining a conservation law and edge features. Our model learns the physics of flows from data by combining bilevel optimization and deep learning. Experiments using traffic and power networks have shown that the proposed model outperforms a set of baselines and learns interpretable physical properties of flow graphs. While we have focused on learning a diagonal regularization matrix, we want to apply our framework to the case of a full matrix. We are also interested in combining different edge measurements in order to learn more complex physical laws, such as described by the fundamental diagram in the LWR model [214, 253, 254, 166].

Chapter 7

Ongoing and Future Work

In this closing section, we first summarize our ongoing project that is motivated by the research developed as part of this thesis, and list promising directions for future research.

7.1 Improving Network Flow Estimation via Knowledge Transfer

One of the main objectives of studying flow graphs is to make accurate flow predictions on edges of a given graph, based on a scarce set of observed edge flows. This problem poses multiple challenges including the limited availability of training data (due to the high cost of flow measurements), as well as the distributional differences between the observed (training) and unobserved (testing) edge flows. Transfer learning is an effective paradigm that can potentially remedy these challenges. One conventional way is to pre-train a model on related tasks where there is more available data, and then fine-tune it on a downstream task of interest, a strategy that showed great success in natural language understanding and computer vision domains.

In this work, we aim to improve flow estimation performance of a model on a given target graph, by leveraging additional data from other flow graphs with similar data distributions [255]. In other words, our goal is to conduct transfer learning across similar flow graphs to collectively make better predictions on each of these flow graphs. Consider the task of predicting traffic flow in LA county traffic network [Silva et al., 2021], where the availability of edges with sensor reading is less than 40% of all the edges. This data scarcity inevitably leads to performance degradation compared to other settings where we have complete data as discussed in [Silva et al., 2021]. To address this challenge, our key idea is to leverage other observations (e.g. sensor readings) from different geographical regions (such as San Francisco, San Diego, Sacramento etc.) to learn a meta-model that can be quickly fine-tuned on a region of interest, *i.e.*, target region. A similar example can be constructed for power transmission networks, where one can leverage available data from various regions/countries and transfer knowledge to a target region based on contextual features (e.g. regional population, line resistances) and network connectivity patterns.

Before we get into more details, we first summarize our progress and contributions. So far:

- We extended our datasets for road traffic networks to include more geographical regions. Specifically, we added the following eight regions in California: San Francisco, Ventura, Riverside, Irvine, Bay Area, Sacramento, San Diego, and Orange County. We further enhanced our feature set by contextual features regarding key regional properties (e.g. residential vs commercial, population).

- We experimented with numerous baseline approaches and identified their potential weaknesses when applied to our problem setting.

- We further proposed a methodology that aims to jointly (i) learn to cluster flow graphs into sub-regions and (ii) meta-learn cluster-specific flow predictors that make predictions within these regions, with both components being learned from a collection of graphs.

7.1.0.1 Problem Setting and Background

Objective: How can we leverage the knowledge extracted from auxiliary graphs to improve semi-supervised flow estimation in the target graph of interest. Before we dive into potential approaches, let us first formally introduce the problem setting and preliminary notations.

Let $\mathcal{G} = \{G_1, G_2, \ldots, G_N\}$ denote a set of N graphs sampled from a task distribution $\mathcal{D}$. For each graph $G(\mathbf{A}, \mathbf{X}, \mathbf{Y})$, we have $\mathbf{A} \in \{0, 1\}^{N \times N}$ as adjacency matrix, $\mathbf{X} \in \mathbb{R}^{|N| \times d}$ as node features, and $\mathbf{Y} \in \mathbb{R}^N$ as node labels. Note that one can also have discrete labels on the nodes such as $\mathbf{Y} = \{y_1, y_2, \ldots, y_m\}$, however, due to the nature of graphs we study in this work, we will primarily focus on the continuous case. We aim to learn a GNN model $f_\theta : \mathbb{R}^d \mapsto \mathbb{R}$ that maps nodes features to observed labels leveraging the graph connections. Typically, the success of the learned model f_θ depends on the availability of node labels. Most real-world graphs have highly sparse observed node labels, which makes the learning of f_θ particularly challenging.

We focus on the learning of f_θ from a collection of graphs $\mathcal{G}$, each with sparse observations. The end goal is to quickly and accurately adapt the learned model to make predictions on never-before-seen graphs, again with limited training data.

This problem is referred as "graph few-shot learning" in the literature, although other referrals also exist such as "learning graphs for knowledge transfer with limited labels" and "transfer learning on graph neural networks". While the few-shot setting explicitly dictates the little amount of available labels, studies on transfer learning also study a similar setting since the knowledge transfer is particularly useful when the available data to learn from is limited. Therefore, these two terms are used almost interchangeably. The methods employed for these two settings also have significant similarities.

We identify two high-level methodologies as potential solutions to this problem.

(i) Conventional pre-training: The goal of pre-training is to (1) learn a good initialization for model parameters (say, θ_0) using the readily available graphs, and (2) later fine-tune the learned model on the target graph of interest with only a few updates. The intuition is that θ_0 captures transferable information across multiple graphs, which makes it easier to adapt to new/unseen graphs. More formally, let $\mathcal{G}^{pre}$ denote the set of pre-training graphs and $\mathcal{G}^{target}$ denote the target graph (or graphs), with $\mathcal{G}^{pre} \cap \mathcal{G}^{\text{target}} = \emptyset$. Target data is split into train/valid/test sets, s.t., $\mathcal{G}^{target} = \{\mathcal{G}^{target}_{train}, \mathcal{G}^{target}_{valid}, \mathcal{G}^{target}_{test}\}$. Formally, the pre-training step computes:

$$\theta_0 = \text{argmin}_{\theta} \mathcal{L}^{pre}(f_\theta, \mathcal{G}^{pre}) \tag{7.1}$$

where $\mathcal{L}^{pre}$ is the pre-training loss function, which may or may not be same as fine-tuning loss $\mathcal{L}^{fine}$. Typically, for classification tasks, it is set to standard cross-entropy loss, while for regression tasks, MSE loss is used. Once pre-training is complete, we next initialize f with θ_0, and further update the model with a few gradient descent steps over $\mathcal{G}^{target}_{train}$. That is, for one step we have:

$$\theta_1 = \theta_0 - \eta \nabla_{\theta_0} \mathcal{L}^{fine}(f_{\theta_0}, \mathcal{G}^{target}_{train}) \tag{7.2}$$

where η is the learning rate. The above update is performed until no improvement is observed on $\mathcal{G}^{target}_{valid}$, while the final performance is measured on held-out test set $\mathcal{G}^{target}_{test}$.

(ii) Meta-learning: In the above two-step paradigm, the pre-training step is decoupled from the fine-tuning step. The learned θ_0 may or may not provide useful knowledge transfer when adapted on a new graph. Negative transfer may occur when the new graph distribution significantly differs from those observed in the pre-training set. Therefore, as an alternative to the pre-training approach, one can employ an **initialization-based meta-learning approach** (such as MAML [Finn et al. (2017)]). The latter approach directly optimizes the models ability to quickly adapt to unseen graphs, rather than optimizing their performance solely on any particular set of graphs as done by the former approach. This paradigm more realistically simulates the fine-tuning (adaptation) process on the target graph. The objective is that one can learn the meta-knowledge across multiple graph instances (also called episodes), which will provide improved adaptability to a target graph.

We now detail how the MAML approach can be employed in our problem by connecting the terminologies used in these two settings. We can treat each graph $G_i \in \mathcal{G}^{pre}$ as a *task*, which consists of *source* and *query* sets that are denoted as $G_i^{\mathcal{S}}$ and $G_i^{\mathcal{Q}}$, respectively. Note that these are analogous to train and test sets defined in the fine-tuning stage (Equation 7.2). Then, during meta-training inner loop, we perform the regular stochastic gradient descent steps on the support loss for each graph (task) G_i. For one step, it is computed as:

$$\theta_i = \theta - \alpha \nabla_\theta \mathcal{L}^{support}(f_\theta, G_i^{\mathcal{S}}) \tag{7.3}$$

where α is the inner step learning rate. Next, the updated parameters of each task are evaluated using the query set during the outer (also called meta-update) step, that is:

$$\theta = \theta - \beta \nabla_\theta \sum_i \mathcal{L}^{query}(f_{\theta_i}, G_i^{\mathcal{Q}}) \tag{7.4}$$

where β is the outer step learning rate (typically smaller than α). We can also re-write the above two-step formulation in a more compact form (for single inner update step) as follows:

$$\theta_* = \text{argmin}_\theta \sum_{G_i \in \mathcal{G}^{pre}} \mathcal{L}^{query}(f_{\theta-\alpha\nabla_\theta \mathcal{L}^{support}(f_\theta, G_i^{\mathcal{S}})}, G_i^{\mathcal{Q}}) \tag{7.5}$$

where θ_* is learned from meta-knowledge across pre-training graphs and is optimal in the sense of adapting to unseen graphs quickly. The meta-testing stage is then analogous to the fine-tuning stage in Equation 7.2, where f_{θ_*} is further updated (adapted) using the training split of target graph ($\mathcal{G}_{train}^{target}$).

Distance metric learning based methods: This family of meta-learning methods approach the problem from a "learning to compare" perspective. This paradigm is successfully applied to multiple machine learning domains. Take image domain as an example, where intuition is that if a model can determine the similarity of two images, it can classify an unseen input image with the labeled instances. Famous distance metric learning based methods include MatchingNet [Vinyals et al. (2016)], ProtoNet [Snell et al. (2017)], and RelationNet [Sung et al. (2018)]. Since originally designed for classification tasks, both MatchingNet and ProtoNet predicts the examples in a query set by comparing the distance between the query feature and the support feature from each class. The MatchingNet computes the average *cosine distance* to examples from each class, while the ProtoNet instead computes *Euclidean distance* to class mean (also referred as *centroids*) of support examples. RelationNet instead replaces the distance measure with a learnable relation module.

The idea of computing label prototypes and further using them to classify query examples are most recently applied in the graph context by [Yao et al. (2020)] and [Huang et al. (2020)]. Both methods aim to capture structural similarity among graphs (or subgraphs), which enables them to form the critical inductive bias for a metric-learning algorithm (such as ProtoNet). Off-the-shelf GNNs are used to encode structural information, which enables direct structural similarity comparison thanks to their expressiveness. The later work further combines the ideas from ProtoNet and MAML to achieve successful results, where each support and query examples correspond to a labeled node and its h-hop immediate neighborhood rather than the entire graph. Both papers show that capturing the "meta-relation" between node representations and labels helps circumvent the issue of limited label information in few-shot settings. Note that this also enables inductive learning, which is crucial for graph few-shot learning since the model is expected to adapt to unseen graphs.

Distance metric learning based methods are also promising candidates for our problem setting. Although these methods originally designed for classification tasks, we can adapt them to our regression-based problem with some modifications. For instance, given examples from query set, instead of comparing them to mean-class centroids computed on the support set (as in ProtoNet), we can directly compare them to support examples in the latent space, and compute the distance-weighted average flow value to use as our flow predictions. Here, the choice (or design) of distance function plays a key role in the success of the model. Our hypothesis is that a suitable choice of distance function may depend on the types of graphs we aim to learn from. Since real-world graphs entail a mix of local and global properties, a distance function that can adaptively account for both aspects is desired.

Consider the problem of learning with traffic networks across different geographical regions, each with sparse flow observations. A central road in Los Angeles might exhibit more similar flow patterns to a central road in San Francisco than a central road in New York City. As an another example, same proteins might take different structural roles in different tissue formations. These nuances can not be captured with a distance function solely defined over local node embeddings that are simply computed through a few layers of graph convolutions. Instead, we need a distance function that can account for such inter-graph variations rooted in data.

7.1.0.2 Proposed Framework

After formally defining our problem and introducing several approaches from the literature as potential solutions, we now propose an alternative approach that is designed to address the aforementioned challenges of transfer learning with flow graphs. We note that although the proposed framework is primarily aimed for flow graphs, investigating its potential on other types of graphs remain a promising future work.

A major motivation behind meta-leaning a graph neural network model across different graphs is that the captured shared meta-knowledge can be used to adapt to any given (and previously unseen) graph. However, this problem is particularly challenging due to the complex nature of graphs with multiple modalities. It is not clear whether such meta-knowledge can be adapted to a new graph in its entirety, as each observed graph may have parts that are unique in nature and are significantly different from the majority of previously observed graphs. It is well-known that flow graphs have heterogeneous structure [5], consisting of multiple sub-regions with potentially varying underlying flow dynamics. We believe aiming to learn a single model to make accurate predictions over the entire graph leads to sub-optimal solutions.

Here, we propose to learn a meta-knowledge in the level of subgraphs (*i.e.* sub-regions), which gives us more control and flexibility in transferring potentially varying knowledge to parts of the target graph. In particular, instead of learning graph-level meta-knowledge that can be adapted to multiple graphs, we propose to jointly break down each graph into clusters and learn cluster-level meta-knowledge that can not only be adapted to different graphs but at the same time, to different parts of the same graph. By doing so, we can learn slightly different adaptations of the shared model across different regions of the graph. With that being our key intuition, we next propose a method that can achieve this end-to-end with all components—including the *clustering component*—being differentiable,

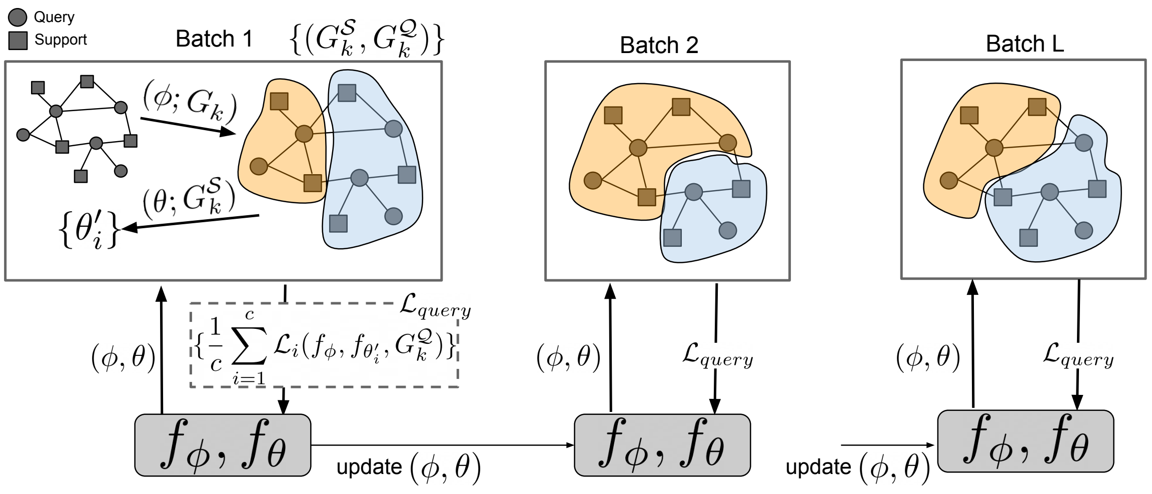

Query
Support
Batch 1
$\{(G_k^{\mathcal{S}}, G_k^{\mathcal{Q}})\}$
$(\phi; G_k)$
$(\theta; G_k^{\mathcal{S}})$
$\{\theta_i'\}$
$\mathcal{L}_{query}$
$\{\frac{1}{c}\sum_{i=1}^{c}\mathcal{L}_i(f_\phi, f_{\theta_i'}, G_k^{\mathcal{Q}})\}$
(ϕ,θ)
f_ϕ, f_θ
update (ϕ,θ)
Batch 2
Batch L

Figure 7.1: An illustration of our proposed method for transfer learning with flow graphs. The proposed method performs joint clustering and intra-cluser flow prediction in an end-to-end fashion using a meta-learning framework. The clustering module f_ϕ captures similar clusters (sub-structures) across graphs, while the flow predictor f_θ is optimized to quickly and accurately adapt to multiple clusters captured by f_ϕ. See text for more details.

hence can be trained with standard neural networks optimization techniques. Figure 7.1 illustrates our proposed approach.

The graph clustering task is naturally unsupervised. The key challenge is to obtain feedback—during training—as to whether the learned clusters are well-fit to our problem setting. To address this challenge, we define a **learnable clustering function** $f_\phi : \mathbb{R}^d \mapsto \mathbb{R}^K$, which takes graph G as input and outputs $\mathbf{C} \in \mathbb{R}^{N\times K}$, *i.e.*, learns cluster assignments for its nodes. Following [256, 257], the design of f_ϕ includes two neural network layers, with the first layer being a GNN layer, and second layer being an MLP, which are followed by a Softmax activation to reach soft-clustering assignments. More formally:

$$f_\phi(G) = Softmax(MLP(GNN(\mathbf{A}, \mathbf{X}))) = \mathbf{C} \tag{7.6}$$

We can now re-define Equation 7.3 to take the form:

$$\theta_i = \theta - \alpha \nabla_\theta \mathcal{L}_i(f_\theta, f_\phi, G^{\mathcal{S}}) \tag{7.7}$$

where $\mathcal{L}$ is the weighted MSE error, where the weights are determined based on the clustering assignments learned by f_ϕ. For cluster i, it is defined as:

$$\mathcal{L}_i(f_\theta, f_\phi, G) = \frac{1}{\|\sqrt{\mathbf{C}_{:,i}}\|} \|(\mathbf{Y} - f_\theta(G)) \odot \sqrt{\mathbf{C}_{:,i}}\|_2^2 \tag{7.8}$$

where $\odot$ represents element-wise vector multiplication, and $\mathbf{C}_{:,i} \in \mathbb{R}^N$ denotes the ith column vector of $\mathbf{C} = f_\phi(G)$. θ_i represents the adapted prediction model parameters based on the assigned clusters of node. Intuitively, nodes with higher clustering assignment scores have more substantial contributions to θ_i. Therefore, we refer θ_i as the cluster-adapted parameters. Equation 7.7 serves as our inner adaptation step, which is repeated in parallel for each cluster $i \in [1, 2, \cdots, K]$. Notice that this step still does not include learning of f_ϕ. The clustering function is instead learned during the outer step, which is formally defined as follows:

$$\begin{aligned} &(\theta, \phi) = (\theta, \phi) - \beta \nabla_{(\theta,\phi)} \mathcal{L}_{outer} \\ &\text{where } \mathcal{L}_{outer}(f_{\theta_i}, f_\phi, G^{\mathcal{Q}}) = \left(\sum_{i=1}^{K} \mathcal{L}_i(f_{\theta_i}, f_\phi, G^{\mathcal{Q}}) \right) + \lambda \mathcal{L}_{rec} \end{aligned} \tag{7.9}$$

In the above formulation, $\mathcal{L}_{rec}$ is the graph reconstruction loss enforcing the cluster assignments to respect the underlying graph structure, similar to [256]. That is, nodes in the same cluster are expected to be strongly connected. λ is a trade-off hyper-parameter, which controls the relative contribution of the two loss terms to our outer objective. We show the formulation for a single graph, while extension to batch computation is straightforward.

Notice that the inner step (7.7) and the outer step (7.9) processes different inputs. While the former step learns cluster-adapted parameters over a support set $G^{\mathcal{S}}$, the latter step jointly evaluates the generalization capability of the learned cluster assignments (parameterized by ϕ) and the predictive model (parameterized by θ) on a query set $G^{\mathcal{Q}}$. The key innovation is that we can jointly learn clustering function during the outer objective, such that the learned function generates clusters that can be better generalized to a collection of graphs. During training, we iterate over a mini-batch of graphs with a certain number of episodes (or epochs), while randomly selecting (non-overlapping) support and query sets from each graph. The number of episodes to train is determined based on the validation performance.

Meta-Testing: During adaptation to a new unseen graph (also called meta-testing), we simply make K many identical copies of θ—denoted as $\{\hat{\theta}_i, i \in [1, 2, \cdots, K]\}$—and run inner steps on each copy independently. We may periodically update ϕ, or not update it at all during this stage. The flow prediction (inference) for each node than can be computed as either the corresponding prediction of the model $f_{\hat{\theta}_i}$ with the highest assignment score, that is, for node n, we choose $f_{\hat{\theta}_i}$ such that $i = \text{argmax}_i \, \mathbf{C_{n,i}}$. Other alternative is to again compute a weighted combination of each model's prediction. The latter approach can be more desirable in case some $\hat{\theta}$s are not well-learned. This case can potentially happen when the available labels for the respective models are either non-existent or highly limited in the new graph.

Discussion: The proposed formulation leads to many critical aspects that one needs to carefully study. First, the clustering problem is known to be hard, and adding the clustering component significantly increases our search space within the optimization landscape. Second, poor cluster assignments (especially early during the training) might lead to undesired performance, while the non-smooth changes in assigned clusters might obstruct convergence. To that end, we identify two orthogonal strategies that we believe can help with more stable learning: (i) pre-training θ without the clustering component, and (ii) updating ϕ less frequently compared to θ, such as once in every 5 epochs. Another potential improvement we can employ is partial weight-tying between f_ϕ and f_θ (such as the first GNN layer), which can reduce the size of our parameter space.

7.1.0.3 Experiments

Synthetic Data

We now create a synthetic experimental setup in order to evaluate our intuitions behind the proposed methodology, as well as to get a quick understanding of its behavior. Our synthetic data includes a graph with 30 nodes, and 3 ground-truth clusters, each with equal sizes of 10 nodes. Our goal is to solve a linear regression task at the node level, which resembles the flow estimation. We assume each cluster has its own underlying model $\theta_i \sim \mathcal{N}(\theta_0, \sigma^2), i \in [1, 2, 3]$ for prediction, while these cluster-specific models are originated from an underlying global graph model (θ_0). Here, d represents the size of the model, while σ is the standard deviation and is set to 0.5 during our experiments. Moreover, each node is assigned random features $x_k \sim \mathcal{N}(0, I_d), k \in [1, 2, \cdots, 30]$ and the ground-truth flow values $y_k = x_k^T \theta_{i^*} \quad \text{s.t. } i^* = \text{argmax}_i C_{k,:}$. Less formally, each ground-truth flow value is assigned based on its features and the corresponding model of the cluster it belongs.

We split nodes in each cluster by 6/2/2 respectively for training (n_{tr}), validation (n_{va}), and testing (n_{te}) sets. We consider two settings: under-parameterized (where d=4) and over-parameterized (where d=12). As the names suggest, the first setting allows to learn accurate models for each cluster (d<n_{tr}), while these cluster-specific models do not encounter enough data points to learn accurate models (d>n_{tr}). We consider two other baselines to compare: *Global* that learns a single global model for the entire graph and

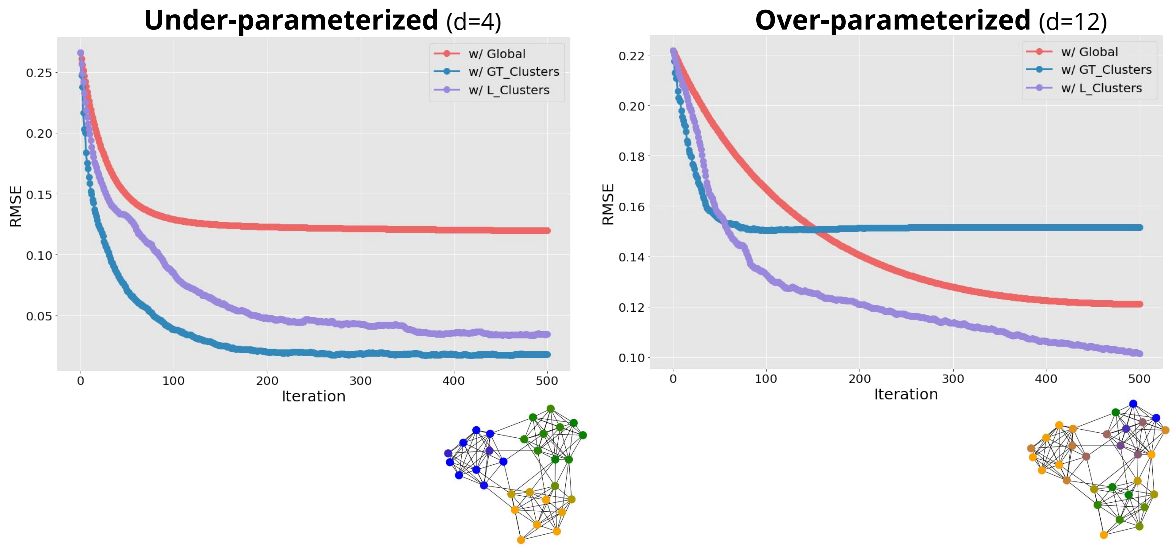


Figure 7.2: Results with three approaches on synthetically generated community network with three underlying clusters. The left part represents the under-parameterized setting, while the right part represents the over-parameterized setting. Performance is measured on the validation set. The proposed method (*L_clusters*, shown in purple) is favored in the latter setting, while it gets closed to the best possible solution (shown in blue) for the former setting. We also plot the clustering assignments learned at the end of training for both cases. See text for more details.

GT_Clusters (stands for ground-truth clusters) that learns three independent models, one for each cluster. Note that the latter variant does not share data points (nodes) across clusters. The proposed variant is denoted by *L_Clusters* and stands for "learned clusters". Our goal is to quantitatively assess the proposed methods joint ability to (i) capture underlying ground-truth clusters and (ii) learn accurate cluster-specific predictors.

Figure 7.2 shows the learning curves of all three models on both settings. The performance is measured by the RMSE metric, and is computed on the testing set. The proposed approach shows desired performance for the over-parameterized case—the setting we are particularly interested in this work—by finding a fair balance between meta-model learning and cluster-specific adaptation. Moreover, the proposed method is also able to learn accurate clustering assignments for the under-parameterized setting, and nears towards the best possible solution where the clustering assignments are known apriori.

Region	Coordinates
San Diego (SD)	[32.5569, 32.8902, -117.2823, -116.9066]
Los Angeles (LA)	[33.8671, 34.2374, -118.5200, -117.8870]
Bay Area (BAY)	[37.2227, 37.5744, -122.3245, -121.7812]
San Francisco (SF)	[37.5647, 37.9557, -122.5727, -122.0477]
Sacramento (SAC)	[38.3599, 38.7406, -121.7456, -121.2287]
Ventura (VEN)	[34.1283, 34.3672, -119.3353, -118.6688]
Irvine (IRV)	[33.4709, 33.7227, -117.9174, -117.5772]
Orange County (ORA)	[33.7262, 33.9048, -118.0879, -117.7463]
Riverside (RIV)	[33.7707, 34.1810, -117.7208, -117.1285]

Table 7.1: Geographical regions in California and their hand-picked coordinates. Each region corresponds to a road network, statistics of which are further summarized in Table 7.2.

Road-Network Flow Estimation

We now shift our focus to a real-world problem that we also used as motivation in the beginning. To re-emphasize, our problem involves estimating flows on a target road network with sparse observations, by applying knowledge transfer from road networks that correspond to other geographical regions. Next, we introduce our dataset curated for this task.

Extended regions and contextual features: We create nine road networks corresponding to the large geographical regions in California, USA. These regions include Los Angeles, San Francisco, Ventura, Riverside, Irvine, Bay Area, Sacramento, San Diego, and Orange County. Geographical boundaries chosen to represent these regions are summarized in Table 7.1. Corresponding road networks are captured from publicly available OpenStreetMap API, while the flows are daily average vehicle counts measured by sensors placed along highways in the year 2018 [1]. We follow the setting in [5] and convert each road network into its corresponding line graph to operate on the node-level, which is more natural to standard GNN architectures. Table 7.2 further summarizes graph statistics for each region, along with the availability of sensor readings within each

[1]Source: `http://pems.dot.ca.gov/`

	Nodes	*Edges*	*Sensors*	*Sensors/Nodes Ratio*
San Diego (SD)	1,704	2,217	623	0.28
Los Angeles (LA)	3,245	4,126	1,578	0.38
Bay Area (BAY)	1,805	2,259	746	0.33
San Francisco (SF)	1,816	2,277	567	0.25
Sacramento (SAC)	994	1,293	515	0.39
Ventura (VEN)	589	756	240	0.31
Irvine (IRV)	689	889	436	0.49
Orange County (OC)	865	1,092	562	0.51
Riverside (RIV)	931	1,247	515	0.41

Table 7.2: Statistics of road networks.

graph. As shown, while graph sizes vary across regions (largest being Los Angeles with 3245 nodes and smallest being Ventura with 589 nodes), the availability of sensors also differ significantly, ranging from 25% (San Francisco) to 51% (Orange County). These statistics demonstrate the complexity of this real-world problem.

In addition to the features used in [5] which include the number of lanes, road type, road length, in and out degree, and centrality of adjacent nodes, we also couple our data with the following extensive list of regional edge features which are also mined from OpenStreetMap API. These additional features include the number of all amenities, food and drink amenities, education amenities, health care amenities, public service amenities, entertainment amenities, all buildings, residential buildings, commercial buildings, all leisure facilities, sports leisure facilities, and nature leisure facilities. These features are collected within a radius mile of each node for all the regions mentioned above. In total, we have 31 features for each node.

Baseline Approaches: We group our baseline approaches into two categories:

(1) *Conventional methods.* These include:

T : Model is trained on target region only, while ignoring other regions.

Joint-T: Model is trained jointly across all the regions. This can be seen as training with one large (global) graph consisting of all the regions.

Pre-T: Model is first pre-trained on source graphs, and then fine-tuned on target graph.

(2) *Meta-Learning approaches.* These consider each graph as a task during an episodic training. These approaches include:

MAML: Model-Agnostic Meta-Learning approach [258]. All GNN layers are adapted in the inner step.

ANIL: Almost No Inner Loop approach [259]. Only the last GNN layer is adapted in the inner step.

MI-GNN: Meta-inductive learning across graphs [260]. Learns a graph-specific transformation of graph features computed as a result of the first GNN layer.

Settings: For each region (seen as the target graph), we conduct experiments with all six baseline approaches. Our choice of GNN architecture is GraphSAGE [250], consisting of two layers with ReLU non-linearity. We split labels of each source graph by 50/50 as support and query sets for meta-training and meta-validation. For adaptation stage (meta-testing), we only use 5% of all available labels on each target graph for training, while the remaining 95% is split evenly for validation and testing. For each target graph, we repeat the experiments 10 times and report the mean and std. Best models are picked based on the validation split of the target graph. Results are reported on the test split of the target graph. For meta-learning approaches, each episode consists of 3 (randomly picked) graphs with 10 inner update steps. For optimization of all methods, we use Adam optimizer with initial learning rate of 0.01.

Metrics: We employ three evaluation metrics that collectively assess the flow estimation performance: Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and Pearson correlation coefficient (Corr).

Results: Table 7.3 summarizes the flow estimation results on five different regions from

Table 7.3: Flow estimation performance of six baselines and proposed approach on five different target regions, corresponding to different counties in California, USA. For the RMSE and MAE metrics, the lower the score the better; while for the Corr metric, the higher the score the better. The best results for each region are bolded.

		T	Joint-T	Pre-T	MI-GNN	MAML	ANIL	Ours
Los Angeles	RMSE	0.0083	0.0085	0.0075	0.0096	0.0073	0.0074	**0.0071**
	MAE	0.0054	0.0057	0.0046	0.0063	0.0045	0.0046	**0.0042**
	Corr	0.7605	0.8136	0.8014	0.6795	0.8279	0.8157	**0.8292**
San Francisco	RMSE	0.0201	0.0107	0.0127	0.0125	0.0112	0.0104	**0.0099**
	MAE	0.0141	0.0072	0.0090	0.0089	0.0074	0.0067	**0.0062**
	Corr	0.4697	0.7665	0.7181	0.6347	0.7420	0.7775	**0.7942**
San Diego	RMSE	0.0156	0.0093	0.0109	0.0135	**0.0083**	0.0086	0.0084
	MAE	0.0108	0.0061	0.0081	0.0106	**0.0058**	0.0059	**0.0058**
	Corr	0.5890	0.8235	0.7506	0.5426	**0.8524**	0.8422	0.8471
Ventura	RMSE	0.0287	0.0187	0.0210	0.0267	0.0163	0.0165	**0.0148**
	MAE	0.0220	0.0119	0.0153	0.0209	0.0108	0.0111	**0.0094**
	Corr	0.6686	0.8271	0.8032	0.5620	0.8476	0.8672	**0.8850**
Riverside	RMSE	0.0198	0.0133	0.0158	0.0163	0.0136	0.0146	**0.0125**
	MAE	0.0145	0.0086	0.0113	0.0122	0.0091	0.0097	**0.0080**
	Corr	0.5206	0.7590	0.6844	0.5973	0.7507	0.7229	**0.7882**

different sizes and rate of available labels (see Table 7.2 for statistics). Our results show that the proposed approach shows significant gains compared to the baseline approaches for four regions: up to 6.6% for Los Angeles, 7.4% for San Francisco, 12.9% for Ventura, and 12.0% for Riverside. Considering the very little amount of training data we have for each of these regions, these improvements demonstrate that the flow estimation task can be solved more effectively by joint clustering and cluster-specific model adaptation. San Diego is the only region where the proposed approach shows similar performance to MAML and ANIL approaches. We note that MAML is a special case of our proposed approach with only one cluster per graph. This suggests that the meta-learned clustering module can not generalize well for the San Diego region. A more detailed investigation is needed to understand the underlying reasons for this outcome.

Our work opens several avenues for future research, which we summarize as follows:

Joint model learning and clustering for other graph families: As part of this thesis, while developing learning methods over graphs, we have primarily focused on flow graphs both from a predictive modeling and a generative modeling perspective. Extending this line of work to other families of graphs (*e.g.*, co-authorship networks, chemical networks) remains a promising future direction to explore.

Designing differentiable clustering modules for graphs: Learning clustering formations directly from data has been an important problem not only for network science but also for many other domains. Although clustering is naturally a self-supervised task, recent advancements in the literature provided solid evidence for the potential of coupling the clustering with a downstream task of interest, such as graph classification [256, 257]. We plan to explore other principled approaches to learning clusters directly from data in more adaptive ways. Some alternatives include (i) employing mean-shift operators over the intermediate latent representations of nodes learned by GNNs, and (ii) dynamically adapting the number of clusters, rather than pre-defining it as a hyper-parameter as done by the previous works.

7.2 Conclusions

Learning problems in the real world often involve complex and richly structured data, coupled with a diverse set of learning objectives depending on the application domain. In this thesis, we have investigated the role of the structure (in which the data forms) towards reaching our learning objectives. Our investigation spans a diverse set of learning problems. These include summarization of (discrete) network processes (Chapter 2), document classification (Chapter 3), personalized item search retrieval (Chapter 4), node/edge-level predictions for complex and attributed graphs (Chapters 6, 7), and data-driven generation of realistic graphs (Chapter 5). For each of these problems, the underlying data of interest resembles different forms of structures including spatial, sequential, and temporal.

Considering the breadth and depth of the problems studied, variety of techniques and computational tools proposed and applied, together with their rigorous analysis and extensive experimental evaluation, our studies in this thesis have led us to the following closing statement:

Building models using structured representations of data (i) allows us to better generalize to the problem domain, and (ii) helps mitigate the effects of data scarcity.

Bibliography

[1] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, *et. al.*, *Relational inductive biases, deep learning, and graph networks*, *arXiv preprint arXiv:1806.01261* (2018).

[2] F. Kocayusufoglu, M. X. Hoang, and A. K. Singh, *Summarizing network processes with network-constrained boolean matrix factorization*, in *2018 IEEE International Conference on Data Mining (ICDM)*, pp. 237–246, IEEE, 2018.

[3] F. Kocayusufoglu, Y. Sheng, N. Vo, J. Wendt, Q. Zhao, S. Tata, and M. Najork, *Riser: Learning better representations for richly structured emails*, in *The World Wide Web Conference*, pp. 886–895, 2019.

[4] H. You, F. Kocayusufoglu, and A. K. Singh, *Danr: Discrepancy-aware network regularization*, in *Proceedings of the 2020 SIAM International Conference on Data Mining*, pp. 208–216, SIAM, 2020.

[5] A. Silva, F. Kocayusufoglu, S. Jafarpour, F. Bullo, A. Swami, and A. Singh, *Combining physics and machine learning for network flow estimation*, .

[6] P. Miettinen *et. al.*, *The discrete basis problem*, *TKDE* (2008).

[7] M. Kitsak *et. al.*, *Identification of influential spreaders in complex networks*, *Nature physics* **6** (2010), no. 11 888.

[8] S. Liu *et. al.*, *Controlling contagion processes in activity driven networks*, *Physical review letters* **112** (2014), no. 11 118702.

[9] A. Meneely *et. al.*, *Predicting failures with developer networks and social network analysis*, in *Proc.16th ACM SIGSOFT*, 2008.

[10] J. Pei *et. al.*, *Closet: An efficient algorithm for mining frequent closed itemsets.*, in *Proc. of ACM SIGMOD DMKD Workshop*, 2000.

[11] R. J. Bayardo Jr, *Efficiently mining long patterns from databases*, *ACM Sigmod Record* **27** (1998), no. 2 85–93.

[12] R. Agrawal *et. al.*, *Mining association rules between sets of items in large databases*, in *Acm sigmod record*, pp. 207–216, ACM, 1993.

[13] J. Vreeken *et. al.*, *Krimp: mining itemsets that compress*, *Data Mining and Knowledge Discovery* **23** (2011), no. 1 169–214.

[14] P. Krajca *et. al.*, *Using frequent closed itemsets for data dimensionality reduction*, in *ICDM*, pp. 1128–1133, IEEE, 2011.

[15] X. Yan and J. Han, *gspan: Graph-based substructure pattern mining*, in *ICDM*, 2002.

[16] X. Yan and J. Han, *Closegraph: mining closed frequent graph patterns*, in *KDD*, 2003.

[17] J. Huan *et. al.*, *Spin: mining maximal frequent subgraphs from graph databases*, in *KDD*, 2004.

[18] L. T. Thomas *et. al.*, *Margin: Maximal frequent subgraph mining*, *TKDD* (2010).

[19] F. Zhu *et. al.*, *Mining colossal frequent patterns by core pattern fusion*, in *ICDE*, pp. 706–715, IEEE, 2007.

[20] P. Miettinen, *Sparse boolean matrix factorizations*, in *ICDM*, 2010.

[21] P. Miettinen and J. Vreeken, *mdl4bmf: Minimum description length for boolean matrix factorization*, *TKDD* (2014).

[22] S. Karaev *et. al.*, *Getting to know the unknown unknowns: Destructive-noise resistant boolean matrix factorization*, in *SIAM*, 2015.

[23] C. Lucchese, S. Orlando, and R. Perego, *Mining top-k patterns from binary datasets in presence of noise*, in *SIAM*, 2010.

[24] C. Lucchese, S. Orlando, and R. Perego, *A unifying framework for mining approximate top-k binary patterns*, *TKDE* (2014).

[25] S. Maurus and C. Plant, *Ternary matrix factorization*, in *ICDM*, 2014.

[26] M. Araujo, P. Ribeiro, and C. Faloutsos, *Faststep: Scalable boolean matrix decomposition*, in *PAKDD*, 2016.

[27] S. Ravanbakhsh, B. Poczos, and R. Greiner, *Boolean matrix factorization and noisy completion via message passing*, *ICML* (2016).

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

[29] J. Leskovec *et. al.*, *Patterns of cascading behavior in large blog graphs*, in *SIAM*, 2007.

[30] N. Barbieri, F. Bonchi, and G. Manco, *Cascade-based community detection*, in *WSDM*, 2013.

[31] U. Feige, *A threshold of ln n for approximating set cover*, *Journal of the ACM (JACM)* **45** (1998), no. 4 634–652.

[32] G. L. Nemhauser *et. al.*, *An analysis of approximations for maximizing submodular set functions*, *Mathematical Programming* (1978).

[33] A.-L. Barabási and R. Albert, *Emergence of scaling in random networks*, *science* **286** (1999), no. 5439 509–512.

[34] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection." `http://snap.stanford.edu/data`, June, 2014.

[35] J. Tang *et. al.*, *Arnetminer: Extraction and mining of academic social networks*, in *KDD'08*, pp. 990–998, 2008.

[36] "Highway traffic network of Los Angeles, CA." `(http://pems.dot.ca.gov)`.

[37] M. Piorkowski *et. al.*, *A parsimonious model of mobile partitioned networks with clustering*, in *COMSNETS, 2009.*

[38] A. Silva *et. al.*, *Prediction-based online trajectory compression*, *arXiv preprint arXiv:1601.06316* (2016).

[39] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts, *Recursive deep models for semantic compositionality over a sentiment treebank*, in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, EMNLP, pp. 1631–1642, 2013.

[40] G. Caruana and M. Li, *A survey of emerging approaches to spam filtering*, *ACM Computing Surveys (CSUR)* **44** (2012), no. 2 9.

[41] M. Najork, *Web spam detection*, in *Encyclopedia of Database Systems*, pp. 3520–3523. Springer, 2009.

[42] K. Shu, A. Sliva, S. Wang, J. Tang, and H. Liu, *Fake news detection on social media: A data mining perspective*, *ACM SIGKDD Explorations Newsletter* **19** (2017), no. 1 22–36.

[43] H. L. Roitblat, A. Kershaw, and P. Oot, *Document categorization in legal electronic discovery: computer classification vs. manual review*, *Journal of the American Society for Information Science and Technology (JASIST)* **61** (2010), no. 1 70–80.

[44] X. Qi and B. D. Davison, *Web page classification: Features and algorithms*, *ACM Computing Surveys (CSUR)* **41** (2009), no. 2 12.

[45] Y. Maarek, *Web mail is not dead!: It's just not human anymore*, in *Proceedings of the 26th International Conference on World Wide Web*, WWW, pp. 5–5, 2017.

[46] M. Grbovic, G. Halawi, Z. Karnin, and Y. Maarek, *How many folders do you really need?: Classifying email into a handful of categories*, in *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, CIKM, pp. 869–878, 2014.

[47] D. Di Castro, Z. Karnin, L. Lewin-Eytan, and Y. Maarek, *You've got mail, and here is what you could do with it!: Analyzing and predicting actions on email messages*, in *Proceedings of the 9th ACM International Conference on Web Search and Data Mining*, WSDM, pp. 307–316, 2016.

[48] Y. Sheng, S. Tata, J. B. Wendt, J. Xie, Q. Zhao, and M. Najork, *Anatomy of a privacy-safe large-scale information extraction system over email*, in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD, pp. 734–743, 2018.

[49] D. Di Castro, I. Gamzu, I. Grabovitch-Zuyev, L. Lewin-Eytan, A. Pundir, N. R. Sahoo, and M. Viderman, *Automated extractions for machine generated mail*, in *Companion Proceedings of the The Web Conference 2018*, WWW, pp. 655–662, 2018.

[50] M. K. Agarwal and J. Singh, *Template trees: Extracting actionable information from machine generated emails*, in *Proceedings of the 30th International Conference on Database and Expert Systems Applications*, DEXA, pp. 3–18, 2018.

[51] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, *Hierarchical attention networks for document classification*, in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL-HLT, pp. 1480–1489, 2016.

[52] Y. Maarek, *Is mail the next frontier in search and data mining?*, in *Proceedings of the 9th ACM International Conference on Web Search and Data Mining*, WSDM, pp. 203–203, 2016.

[53] P. Le Hégaret, R. Whitmer, and L. Wood, "Document object model (dom)." `http://www.w3.org/DOM`, 2005.

[54] Google, "Analyzing entities." `https://cloud.google.com/natural-language/docs/analyzing-entities`, 2019.

[55] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernández, M. Kay, J. Robie, and J. Siméon, *XML path language (xpath)*, *World Wide Web Consortium (W3C)* (2007).

[56] D. Cai, S. Yu, J.-R. Wen, and W.-Y. Ma, *Extracting content structure for web pages based on visual representation*, in *Asia-Pacific Web Conference*, pp. 406–417, Springer, 2003.

[57] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, *Neural computation* **9** (1997), no. 8 1735–1780.

[58] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, *Distributed representations of words and phrases and their compositionality*, in *Proceedings of the 26th International Conference on Neural Information Processing Systems*, NIPS, pp. 3111–3119, 2013.

[59] J. Pennington, R. Socher, and C. Manning, *Glove: Global vectors for word representation*, in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, EMNLP, pp. 1532–1543, 2014.

[60] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, *Enriching word vectors with subword information*, *arXiv preprint arXiv:1607.04606* (2016).

[61] R. V. Guha, D. Brickley, and S. Macbeth, *Schema. org: evolution of structured data on the web*, *Communications of the ACM* **59** (2016), no. 2 44–51.

[62] N. Ailon, Z. S. Karnin, E. Liberty, and Y. Maarek, *Threading machine generated email*, in *Proceedings of the 6th ACM International Conference on Web Search and Data Mining*, WSDM, pp. 405–414, 2013.

[63] N. Avigdor-Elgrabli, M. Cwalinski, D. Di Castro, I. Gamzu, I. Grabovitch-Zuyev, L. Lewin-Eytan, and Y. Maarek, *Structural clustering of machine-generated mail*, in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM, pp. 217–226, 2016.

[64] Y. Liu and M. Lapata, *Learning structured text representations*, *Transactions of the Association of Computational Linguistics* **6** (2018) 63–75.

[65] J. Duchi, E. Hazan, and Y. Singer, *Adaptive subgradient methods for online learning and stochastic optimization*, *Journal of Machine Learning Research (JMLR)* **12** (2011), no. Jul 2121–2159.

[66] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, *arXiv preprint arXiv:1412.6980* (2014).

[67] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley, *Google Vizier: A service for black-box optimization*, in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD, pp. 1487–1495, 2017.

[68] A. McCallum, K. Nigam, *et. al.*, *A comparison of event models for naive bayes text classification*, in *AAAI-98 Workshop on Learning for Text Categorization*, vol. 752, pp. 41–48, 1998.

[69] Y. Yang and X. Liu, *A re-examination of text categorization methods*, in *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR, pp. 42–49, 1999.

[70] B. Pang, L. Lee, and S. Vaithyanathan, *Thumbs up?: sentiment classification using machine learning techniques*, in *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, EMNLP, pp. 79–86, 2002.

[71] T. Mikolov, K. Chen, G. Corrado, and J. Dean, *Efficient estimation of word representations in vector space*, *arXiv preprint arXiv:1301.3781* (2013).

[72] Q. Le and T. Mikolov, *Distributed representations of sentences and documents*, in *Proceedings of the 31st International Conference on Machine Learning*, ICML, pp. 1188–1196, 2014.

[73] J. Li, M.-T. Luong, and D. Jurafsky, *A hierarchical neural autoencoder for paragraphs and documents*, *arXiv preprint arXiv:1506.01057* (2015).

[74] S. Lai, L. Xu, K. Liu, and J. Zhao, *Recurrent convolutional neural networks for text classification*, in *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, AAAI, pp. 2267–2273, 2015.

[75] D. Tang, B. Qin, and T. Liu, *Document modeling with gated recurrent neural network for sentiment classification*, in *Proceedings of the 2015 conference on empirical methods in natural language processing*, pp. 1422–1432, 2015.

[76] K. S. Tai, R. Socher, and C. D. Manning, *Improved semantic representations from tree-structured long short-term memory networks*, *arXiv preprint arXiv:1503.00075* (2015).

[77] J. Bradbury, S. Merity, C. Xiong, and R. Socher, *Quasi-recurrent neural networks*, *arXiv preprint arXiv:1611.01576* (2016).

[78] X. Zhang, J. Zhao, and Y. LeCun, *Character-level convolutional networks for text classification*, in *Proceedings of the 28th International Conference on Neural Information Processing Systems*, NIPS, pp. 649–657, 2015.

[79] Y. Kim, *Convolutional neural networks for sentence classification*, *arXiv preprint arXiv:1408.5882* (2014).

[80] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, *arXiv preprint arXiv:1409.0473* (2014).

[81] W. C. Mann and S. A. Thompson, *Rhetorical structure theory: Toward a functional theory of text organization*, *Text-Interdisciplinary Journal for the Study of Discourse* **8** (1988), no. 3 243–281.

[82] P. Bhatia, Y. Ji, and J. Eisenstein, *Better document-level sentiment analysis from rst discourse parsing*, in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 2212–2218, 2015.

[83] Y. Ji and N. Smith, *Neural discourse structure for text categorization*, *arXiv preprint arXiv:1702.01829* (2017).

[84] R. Bekkerman, A. McCallum, and G. Huang, *Automatic categorization of email into folders: Benchmark experiments on Enron and SRI corpora*, Center for Intelligent Information Retrieval report IR-418, University of Massachusetts, 2005.

[85] G. V. Cormack, *Email spam filtering: A systematic review*, *Foundations and Trends® in Information Retrieval* **1** (2008), no. 4 335–455.

[86] D. Aberdeen, O. Pacovsky, and A. Slater, *The learning behind Gmail priority inbox*, in *LCCC: NIPS 2010 Workshop on Learning on Cores, Clusters and Clouds*, 2010.

[87] S. Wu, L. Hsiao, X. Cheng, B. Hancock, T. Rekatsinas, P. Levis, and C. Ré, *Fonduer: Knowledge base construction from richly formatted data*, in *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD, pp. 1301–1316, 2018.

[88] A. Anderson *et. al.*, *Algorithmic effects on the diversity of consumption on spotify*, in *Proceedings of TheWebConf'20*, pp. 2155–2165, 2020.

[89] J. Burgess and J. Green, *YouTube: Online video and participatory culture.* John Wiley & Sons, 2018.

[90] C. Eksombatchai *et. al.*, *Pixie: A system for recommending 3+ billion items to 200+ million users in real-time*, in *Proceedings of TheWebConf'18*, 2018.

[91] W. W. Moe, *Buying, searching, or browsing: Differentiating between online shoppers using in-store navigational clickstream*, *Journal of consumer psychology* **13** (2003), no. 1-2 29–39.

[92] P. Sondhi *et. al.*, *A taxonomy of queries for e-commerce search*, in *Proceedings of ACM SIGIR'18*, pp. 1245–1248, 2018.

[93] N. Su, J. He, Y. Liu, M. Zhang, and S. Ma, *User intent, behaviour, and perceived satisfaction in product search*, in *Proceedings of ACM WSDM'18*, pp. 547–555, 2018.

[94] Q. Ai, Y. Zhang, K. Bi, X. Chen, and W. B. Croft, *Learning a hierarchical embedding model for personalized product search*, in *Proceedings of ACM SIGIR'17*, pp. 645–654, 2017.

[95] Y. Guo *et. al.*, *Attentive long short-term preference modeling for personalized product search*, *ACM TOIS'19* **37** (2019), no. 2 1–27.

[96] Q. Ai, D. N. Hill, S. Vishwanathan, and W. B. Croft, *A zero attention model for personalized product search*, in *Proceedings of ACM CIKM'19*, pp. 379–388, 2019.

[97] N. Matthijs and F. Radlinski, *Personalizing web search using long term browsing history*, in *Proceedings of ACM WSDM'11*, pp. 25–34, 2011.

[98] D. Sontag, K. Collins-Thompson, P. N. Bennett, R. W. White, S. Dumais, and B. Billerbeck, *Probabilistic models for personalizing web search*, in *Proceedings of ACM WSDM'12*, pp. 433–442, 2012.

[99] J. Vosecky, K. W.-T. Leung, and W. Ng, *Collaborative personalized twitter search with topic-language models*, in *Proceedings of ACM SIGIR'14*, pp. 53–62, 2014.

[100] P. Covington, J. Adams, and E. Sargin, *Deep neural networks for youtube recommendations*, in *Proceedings of ACM RecSys'16*, pp. 191–198, 2016.

[101] S. Lamkhede and S. Das, *Challenges in search on streaming services: netflix case study*, in *Proceedings of ACM SIGIR'19*, pp. 1371–1374, 2019.

[102] A. Vaswani *et. al.*, *Attention is all you need*, in *NeurIPS'17*, 2017.

[103] J. Devlin *et. al.*, *Bert: Pre-training of deep bidirectional transformers for language understanding*, *arXiv preprint arXiv:1810.04805* (2018).

[104] C.-C. Chiu *et. al.*, *State-of-the-art speech recognition with sequence-to-sequence models*, in *ICASSP'18*, pp. 4774–4778, IEEE, 2018.

[105] W.-C. Kang and J. McAuley, *Self-attentive sequential recommendation*, in *ICDM'18*, pp. 197–206, IEEE, 2018.

[106] P. G. Campos, F. Díez, and I. Cantador, *Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols*, *User Modeling and User-Adapted Interaction* **24** (2014), no. 1 67–119.

[107] Y. Zhu *et. al.*, *What to do next: Modeling user behaviors by time-lstm.*, in *IJCAI'17*, vol. 17, pp. 3602–3608, 2017.

[108] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan, *Self-attention with functional time representation learning*, in *NeurIPS'19*, 2019.

[109] J. Li, Y. Wang, and J. McAuley, *Time interval aware self-attention for sequential recommendation*, in *Proceedings of WSDM'20*, pp. 322–330, 2020.

[110] W. Ji *et. al.*, *Sequential recommender via time-aware attentive memory network*, in *Proceedings of ACM CIKM'20*, pp. 565–574, 2020.

[111] J. Wu, R. Cai, and H. Wang, *Déjà vu: A contextualized temporal attention mechanism for sequential recommendation*, in *Proceedings of The Web Conference 2020*, pp. 2199–2209, 2020.

[112] J.-Y. Jiang, T. Wu, G. Roumpos, H.-T. Cheng, X. Yi, E. Chi, H. Ganapathy, N. Jindal, P. Cao, and W. Wang, *End-to-end deep attentive personalized item retrieval for online content-sharing platforms*, in *Proceedings of TheWebConf'20*, pp. 2870–2877, 2020.

[113] N. Sachdeva and J. McAuley, *How useful are reviews for recommendation? a critical review and potential improvements*, in *Proceedings of ACM SIGIR'20*, pp. 1845–1848, 2020.

[114] R. Datta, D. Joshi, J. Li, and J. Z. Wang, *Image retrieval: Ideas, influences, and trends of the new age*, *ACM Computing Surveys (Csur)* **40** (2008), no. 2 1–60.

[115] A. Goswami, N. Chittar, and C. H. Sung, *A study on the impact of product images on user clicks for online shopping*, in *Proceedings of TheWebConf'11*, pp. 45–46, 2011.

[116] T. Xiao, J. Ren, Z. Meng, H. Sun, and S. Liang, *Dynamic bayesian metric learning for personalized product search*, in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 1693–1702, 2019.

[117] X. Bu, J. Zhu, X. Qian, and M. IEEE, *Personalized product search based on user transaction history and hypergraph learning*, *Multimedia Tools and Applications* **79** (2020) 22157–22175.

[118] S. Liu, W. Gu, G. Cong, and F. Zhang, *Structural relationship representation learning with graph embedding for personalized product search*, in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 915–924, 2020.

[119] K. Bi, Q. Ai, and W. B. Croft, *Learning a fine-grained review-based transformer model for personalized product search*, in *Proceedings of ACM SIGIR'21*, pp. 123–132, 2021.

[120] M. Grbovic and H. Cheng, *Real-time personalization using embeddings for search ranking at airbnb*, in *Proceedings of ACM SIGKDD'18*, pp. 311–320, 2018.

[121] X. Chen *et. al.*, *Sequential recommendation with user memory networks*, in *Proceedings of ACM WSDM'18*, pp. 108–116, 2018.

[122] C. Ma, P. Kang, and X. Liu, *Hierarchical gating networks for sequential recommendation*, in *Proceedings of ACM SIGKDD'19*, pp. 825–833, 2019.

[123] P. Shaw, J. Uszkoreit, and A. Vaswani, *Self-attention with relative position representations*, in *Proceedings of NAACL'18*, pp. 464–468, 2018.

[124] Y. Koren, *Collaborative filtering with temporal dynamics*, in *Proceedings ACM SIGKDD'09*, pp. 447–456, 2009.

[125] Y. Li, N. Du, and S. Bengio, *Time-dependent representation for neural event sequence prediction*, *arXiv preprint arXiv:1708.00065* (2017).

[126] G. Lai, W.-C. Chang, Y. Yang, and H. Liu, *Modeling long-and short-term temporal patterns with deep neural networks*, in *Proceedings of ACM SIGIR'18*, pp. 95–104, 2018.

[127] S. Hirsch, I. Guy, A. Nus, A. Dagan, and O. Kurland, *Query reformulation in e-commerce search*, in *Proceedings of ACM SIGIR'20*, 2020.

[128] F. Sun *et. al.*, *Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer*, in *Proceedings of ACM CIKM'19*, pp. 1441–1450, 2019.

[129] I. Vulić and M.-F. Moens, *Monolingual and cross-lingual information retrieval models based on (bilingual) word embeddings*, in *Proceedings of ACM SIGIR'15*, pp. 363–372, 2015.

[130] H. Mei and J. M. Eisner, *The neural hawkes process: A neurally self-modulating multivariate point process*, *Advances in Neural Information Processing Systems* **30** (2017).

[131] J. Ni, J. Li, and J. McAuley, *Justifying recommendations using distantly-labeled reviews and fine-grained aspects*, in *Proceedings of EMNLP-IJCNLP'19*, pp. 188–197, 2019.

[132] C. Van Gysel, M. de Rijke, and E. Kanoulas, *Learning latent vector spaces for product search*, in *CIKM'16*, pp. 165–174, 2016.

[133] J. Rowley, *Product search in e-shopping: a review and research propositions*, *Journal of consumer marketing* (2000).

[134] J. Huang *et. al.*, *Improving sequential recommendation with knowledge-enhanced memory networks*, in *Proceedings of ACM SIGIR'18*, pp. 505–514, 2018.

[135] R. Albert and A.-L. Barabási, *Statistical mechanics of complex networks*, *Reviews of modern physics* **74** (2002), no. 1 47.

[136] P. Erdös and A. Rényi, *On random graphs*, *Publicationes mathematicae* **6** (1959), no. 26 290–297.

[137] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann, *Netgan: Generating graphs via random walks*, in *ICML*, 2018.

[138] H. Dai, A. Nazi, Y. Li, B. Dai, and D. Schuurmans, *Scalable deep generative modeling for sparse graphs*, in *International Conference on Machine Learning*, pp. 2302–2312, PMLR, 2020.

[139] A. Grover, A. Zweig, and S. Ermon, *Graphite: Iterative generative modeling of graphs*, in *ICML*, 2019.

[140] S. Li, S. Xiao, S. Zhu, N. Du, Y. Xie, and L. Song, *Learning temporal point processes via reinforcement learning*, in *NeurIPS'18*, 2018.

[141] R. Liao, Y. Li, Y. Song, S. Wang, W. Hamilton, D. K. Duvenaud, R. Urtasun, and R. Zemel, *Efficient graph generation with graph recurrent attention networks*, in *NeurIPS*, 2019.

[142] J. Liu, A. Kumar, J. Ba, J. Kiros, and K. Swersky, *Graph normalizing flows*, in *NeurIPS*, 2019.

[143] C. Niu, Y. Song, J. Song, S. Zhao, A. Grover, and S. Ermon, *Permutation invariant graph generation via score-based generative modeling*, in *International Conference on Artificial Intelligence and Statistics*, pp. 4474–4484, PMLR, 2020.

[144] M. Simonovsky and N. Komodakis, *Graphvae: Towards generation of small graphs using variational autoencoders*, in *ICANN*, 2018.

[145] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec, *Graphrnn: Generating realistic graphs with deep auto-regressive models*, in *ICML*, 2018.

[146] W. Jin, R. Barzilay, and T. Jaakkola, *Junction tree variational autoencoder for molecular graph generation*, in *ICML*, 2018.

[147] C. Shi, M. Xu, Z. Zhu, W. Zhang, M. Zhang, and J. Tang, *Graphaf: a flow-based autoregressive model for molecular graph generation*, in *ICLR*, 2020.

[148] X. Zheng, B. Aragam, *et. al.*, *Dags with no tears: Continuous optimization for structure learning*, in *NeurIPS*, 2018.

[149] H. Chu, D. Li, *et. al.*, *Neural turtle graphics for modeling city road layouts*, in *ICCV*, 2019.

[150] M. Brockschmidt, M. Allamanis, A. L. Gaunt, and O. Polozov, *Generative code modeling with graphs*, in *ICLR*, 2019.

[151] F. Calabrese, G. Di Lorenzo, *et. al.*, *Estimating origin-destination flows using mobile phone location data*, *IEEE Pervasive Computing* (2011), no. 4 36–44.

[152] A. Bassolas, H. Barbosa-Filho, B. Dickinson, X. Dotiwalla, P. Eastham, R. Gallotti, G. Ghoshal, B. Gipson, S. A. Hazarie, H. Kautz, *et. al.*, *Hierarchical organization of urban mobility and its connection with city livability*, *Nature communications* **10** (2019), no. 1 1–10.

[153] J. Jia, M. T. Schaub, S. Segarra, and A. R. Benson, *Graph-based semi-supervised & active learning for edge flows*, in *KDD*, 2019.

[154] M. Chinazzi, J. T. Davis, *et. al.*, *The effect of travel restrictions on the spread of the 2019 novel coronavirus (covid-19) outbreak*, *Science* **368** (2020), no. 6489 395–400.

[155] F. Dörfler and F. Bullo, *Synchronization in complex networks of phase oscillators: A survey*, *Automatica* **50** (2014), no. 6 1539–1564.

[156] D. Hill and G. Chen, *Power systems as dynamic networks*, in *2006 IEEE International Symposium on Circuits and Systems*, 2006.

[157] J. D. Orth, I. Thiele, and B. Palsson, *What is flux balance analysis?*, *Nat Biotechnol* **28** (Mar, 2010) 245–248.

[158] D. J. Watts and S. H. Strogatz, *Collective dynamics of âsmall-worldânetworks*, *Nature* **393** (1998), no. 6684 440–442.

[159] T. N. Kipf and M. Welling, *Semi-supervised classification with graph convolutional networks*, in *ICLR*, 2017.

[160] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, *Graph attention networks*, in *ICLR*, 2017.

[161] C. Yang, P. Zhuang, W. Shi, A. Luu, and P. Li, *Conditional structure generation through graph variational generative adversarial nets*, in *NeurIPS*, 2019.

[162] D. P. Kingma and M. Welling, *Auto-encoding variational bayes*, *arXiv preprint arXiv:1312.6114* (2013).

[163] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative adversarial nets*, in *NeurIPS*, 2014.

[164] Y. Song and S. Ermon, *Generative modeling by estimating gradients of the data distribution*, in *Proceedings of the 33rd Annual Conference on Neural Information Processing Systems*, 2019.

[165] A. Bressan, S. Čanić, M. Garavello, M. Herty, and B. Piccoli, *Flows on networks: recent results and perspectives*, *EMS Surveys in Mathematical Sciences* **1** (2014), no. 1 47–111.

[166] M. Garavello and B. Piccoli, *Traffic Flow on Networks: Conservation Laws Model.* AIMS series on applied mathematics. AIMS, 2006.

[167] M. Belkin and P. Niyogi, *Laplacian eigenmaps for dimensionality reduction and data representation*, *Neural computation* **15** (2003), no. 6 1373–1396.

[168] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, *Gated graph sequence neural networks*, in *ICLR*, 2016.

[169] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in *CVPR*, 2016.

[170] M. Arjovsky, S. Chintala, and L. Bottou, *Wasserstein generative adversarial networks*, in *ICML*, 2017.

[171] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, *Improved training of wasserstein gans*, in *NeurIPS*, 2017.

[172] J. Hörsch, F. Hofmann, D. Schlachtberger, and T. Brown, *Pypsa-eur: An open optimisation model of the european transmission system*, *Energy strategy reviews* **22** (2018) 207–215.

[173] L. Theis, A. v. d. Oord, and M. Bethge, *A note on the evaluation of generative models*, *arXiv preprint arXiv:1511.01844* (2015).

[174] D. J. Aldous, *Lower bounds for covering times for reversible markov chains and random walks on graphs*, *Journal of Theoretical Probability* **2** (1989), no. 1 91–100.

[175] N. Taxi and L. commission, *TLC Trip Record Data*, 2019. Available at `https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page`. Accessed: December 2019.

[176] R. Kinney, P. Crucitti, R. Albert, and V. Latora, *Modeling cascading failures in the north american power grid*, *The European Physical Journal B-Condensed Matter and Complex Systems* **46** (2005), no. 1 101–107.

[177] H. W. Dommel and W. F. Tinney, *Optimal power flow solutions*, *IEEE Transactions on power apparatus and systems* (1968), no. 10 1866–1876.

[178] F. Dörfler, J. W. Simpson-Porco, and F. Bullo, *Electrical networks and algebraic graph theory: Models, properties, and applications*, *Proceedings of the IEEE* **106** (2018), no. 5 977–1005.

[179] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, *Neural message passing for quantum chemistry*, in *ICML*, 2017.

[180] A. Srivastava, L. Valkov, C. Russell, M. U. Gutmann, and C. Sutton, *Veegan: Reducing mode collapse in gans using implicit variational learning*, in *Advances in neural information processing systems*, pp. 3308–3318, 2017.

[181] M. E. Newman, *Mixing patterns in networks*, *Physical review E* **67** (2003), no. 2 026126.

[182] D. Hallac *et. al.*, *Network lasso: Clustering and optimization in large graphs*, in *ACM SIGKDD*, 2015.

[183] Y.-X. Wang, J. Sharpnack, A. J. Smola, and R. J. Tibshirani, *Trend filtering on graphs*, in *AISTATS*, 2015.

[184] J. Pang *et. al.*, *Graph laplacian regularization for image denoising*, *IEEE Transactions on Image Processing* (2017).

[185] C. Li and H. Li, *Network-constrained regularization and variable selection for analysis of genomic data*, *Bioinformatics* **24** (2008), no. 9 1175–1182.

[186] T. D. Hocking *et. al.*, *Clusterpath an algorithm for clustering using convex fusion penalties*, in *ICML*, 2011.

[187] E. C. Chi *et. al.*, *Splitting methods for convex clustering*, *Journal of Computational and Graphical Statistics* **24** (2015), no. 4.

[188] Y.-Y. Liu, J.-J. Slotine, and A.-L. Barabási, *Controllability of complex networks*, *Nature* **473** (2011), no. 7346 167.

[189] L. Anselin, *Issues in the specification and interpretation of spatial regression models*, *Agricultural economics* **27** (2002), no. 3.

[190] R. Harris, J. Moffat, and V. Kravtsova, *In search of 'w'*, *Spatial Economic Analysis* **6** (2011), no. 3 249–270.

[191] M. Kim and J. Leskovec, *Nonparametric multi-group membership model for dynamic networks*, in *NIPS*, 2013.

[192] Q. Ho *et. al.*, *Evolving cluster mixed-membership blockmodel for time-evolving networks*, in *AISTATS*, 2011.

[193] M. Belkin, P. Niyogi, and V. Sindhwani, *Manifold regularization: A geometric framework for learning from labeled and unlabeled examples*, *Journal of machine learning research* **7** (2006), no. Nov 2399–2434.

[194] M. Belkin *et. al.*, *Regularization and semi-supervised learning on large graphs*, in *COLT*, 2004.

[195] N. Guan, D. Tao, Z. Luo, and B. Yuan, *Manifold regularized discriminative nonnegative matrix factorization with fast gradient descent*, *IEEE Transactions on Image Processing* **20** (2011), no. 7 2030–2048.

[196] K. Q. Weinberger, F. Sha, Q. Zhu, and L. K. Saul, *Graph laplacian regularization for large-scale semidefinite programming*, in *NIPS*, 2007.

[197] F. Lindsten *et. al.*, *Clustering using sum-of-norms regularization*, in *Statistical Signal Processing Workshop*, 2011.

[198] R. Tibshirani *et. al.*, *Sparsity and smoothness via the fused lasso*, *Journal of the Royal Statistical Society* (2005), no. 1.

[199] L. I. Rudin, S. Osher, and E. Fatemi, *Nonlinear total variation based noise removal algorithms*, *Physica D: nonlinear phenomena* **60** (1992), no. 1-4.

[200] H.-F. Yu *et. al.*, *Temporal regularized matrix factorization for high-dimensional time series prediction*, in *NIPS*, 2016.

[201] J. E. Vogt and V. Roth, *A complete analysis of the $\ell_{1,p}$ group-lasso*, in *ICML*, 2012.

[202] D. Gabay and B. Mercier, *A dual algorithm for the solution of nonlinear variational problems via finite element approximation*, *Computers & Mathematics with Applications* no. 1.

[203] Y. Peng *et. al.*, *Robust alignment by sparse and low-rank decomposition for linearly correlated images*, *PAMI* (2012).

[204] S. Boyd *et. al.*, *Distributed optimization and statistical learning via the alternating direction method of multipliers*, *Foundations and Trends® in Machine Learning* **3** (2011), no. 1.

[205] J. Eckstein *et. al.*, *On the douglas—rachford splitting method and the proximal point algorithm for maximal monotone operators*, *Mathematical Programming* **55** (1992), no. 1.

[206] R. Nishihara *et. al.*, *A general analysis of the convergence of admm*, in *ICML*, pp. 343–352, 2015.

[207] B. He and X. Yuan, *On non-ergodic convergence rate of douglas–rachford alternating direction method of multipliers*, *Numerische Mathematik* **130** (2015), no. 3 567–577.

[208] D. Hallac, Y. Park, S. Boyd, and J. Leskovec, *Network inference via the time-varying graphical lasso*, 2017.

[209] H. Wang, F. Nie, and H. Huang, *Low-rank tensor completion with spatio-temporal consistency*, in *AAAI*, 2014.

[210] X.-H. Dang *et. al.*, *Subnetwork mining with spatial and temporal smoothness*, in *SDM*, 2017.

[211] P. Gong, J. Ye, and C. Zhang, *Robust multi-task feature learning*, in *ACM SIGKDD*, 2012.

[212] J. Xu *et. al.*, *Factorized multi-task learning for task discovery in personalized medical models*, in *SDM*, 2015.

[213] P. A. Soranno *et. al.*, *Lagos-ne*, *GigaScience* **6** (2017), no. 12.

[214] M. J. Lighthill and G. B. Whitham, *On kinematic waves ii. a theory of traffic flow on long crowded roads*, *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences* **229** (1955), no. 1178 317–345.

[215] P. I. Richards, *Shock waves on the highway*, *Operations research* **4** (1956), no. 1 42–51.

[216] Y. Li, R. Yu, C. Shahabi, and Y. Liu, *Diffusion convolutional recurrent neural network: Data-driven traffic forecasting*, *arXiv preprint arXiv:1707.01926* (2017).

[217] B. Yu, H. Yin, and Z. Zhu, *Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting*, in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pp. 3634–3640, 2018.

[218] J. Jia, M. T. Schaub, S. Segarra, and A. R. Benson, *Graph-based semi-supervised and active learning for edge flows*, in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, p. 761–771, 2019.

[219] L. Franceschi, M. Donini, P. Frasconi, and M. Pontil, *Forward and reverse gradient-based hyperparameter optimization*, in *ICML*, vol. 70, pp. 1165–1173, JMLR, 2017.

[220] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, and A. Ramadhan, *Universal differential equations for scientific machine learning*, *arXiv preprint arXiv:2001.04385* (2020).

[221] J. Hanc, S. Tuleja, and M. Hancova, *Symmetries and conservation laws: Consequences of noether's theorem*, *American Journal of Physics* **72** (2004), no. 4 428–435.

[222] A. Tarantola, *Inverse problem theory and methods for model parameter estimation*, vol. 89. SIAM, 2005.

[223] S. Arridge, P. Maass, O. Öktem, and C.-B. Schönlieb, *Solving inverse problems using data-driven models*, *Acta Numerica* **28** (2019) 1–174.

[224] G. M. Coclite, M. Garavello, and B. Piccoli, *Traffic flow on a road network*, *SIAM Journal on Mathematical Analysis* **36** (2005), no. 6 1862–1886.

[225] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows*, .

[226] B. Colson, P. Marcotte, and G. Savard, *An overview of bilevel optimization*, *Annals of operations research* **153** (2007), no. 1 235–256.

[227] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil, *Bilevel programming for hyperparameter optimization and meta-learning*, in *ICML*, vol. 80, pp. 1563–1572, PMLR (Proceedings of Machine Learning Research), 2018.

[228] J. Lorraine, P. Vicol, and D. Duvenaud, *Optimizing millions of hyperparameters by implicit differentiation*, in *International Conference on Artificial Intelligence and Statistics*, pp. 1540–1552, PMLR, 2020.

[229] G. Cybenko, *Approximation by superpositions of a sigmoidal function*, *Mathematics of control, signals and systems* **2** (1989), no. 4 303–314.

[230] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, *How powerful are graph neural networks?*, in *ICLR*, 2018.

[231] M. Defferrard, X. Bresson, and P. Vandergheynst, *Convolutional neural networks on graphs with fast localized spectral filtering*, in *NeurIPS*, 2016.

[232] D. K. Hammond, P. Vandergheynst, and R. Gribonval, *Wavelets on graphs via spectral graph theory*, *Applied and Computational Harmonic Analysis* **30** (2011), no. 2 129–150.

[233] L. Bottou and O. Bousquet, *The tradeoffs of large scale learning*, in *Advances in neural information processing systems*, pp. 161–168, 2008.

[234] J. Domke, *Generic methods for optimization-based modeling*, in *Artificial Intelligence and Statistics*, pp. 318–326, 2012.

[235] T. Brown, J. Hörsch, and D. Schlachtberger, *Pypsa: Python for power system analysis*, *arXiv preprint arXiv:1707.09913* (2017).

[236] J. C. Herrera, D. B. Work, R. Herring, X. J. Ban, Q. Jacobson, and A. M. Bayen, *Evaluation of traffic data obtained via GPS-enabled mobile phones: The mobile century field experiment*, *Transportation Research Part C: Emerging Technologies* **18** (2010), no. 4 568–583.

[237] D. B. Work, S. Blandin, O.-P. Tossavainen, B. Piccoli, and A. M. Bayen, *A traffic model for velocity data assimilation*, *Applied Mathematics Research eXpress* **2010** (2010), no. 1 1–35.

[238] L. R. Ford Jr and D. R. Fulkerson, *Flows in networks.* Princeton university press, 2015.

[239] H. W. Engl, M. Hanke, and A. Neubauer, *Regularization of inverse problems*, vol. 375. Springer Science & Business Media, 1996.

[240] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, *Journal of Computational Physics* **378** (2019) 686–707.

[241] Z. Long, Y. Lu, X. Ma, and B. Dong, *Pde-net: Learning pdes from data*, in *35th International Conference on Machine Learning, ICML 2018*, pp. 5067–5078, International Machine Learning Society (IMLS), 2018.

[242] Y. Bengio, *Gradient-based optimization of hyperparameters*, *Neural computation* **12** (2000), no. 8 1889–1900.

[243] J. Larsen, L. K. Hansen, C. Svarer, and M. Ohlsson, *Design and regularization of neural networks: the optimal use of a validation set*, in *Neural Networks for Signal Processing VI. Proceedings of the 1996 IEEE Signal Processing Society Workshop*, pp. 62–71, IEEE, 1996.

[244] D. Maclaurin, D. Duvenaud, and R. Adams, *Gradient-based hyperparameter optimization through reversible learning*, in *International Conference on Machine Learning*, pp. 2113–2122, 2015.

[245] F. Pedregosa, *Hyperparameter optimization with approximate gradient*, in *Proceedings of the 33rd International Conference on International Conference on Machine Learning-Volume 48*, pp. 737–746, 2016.

[246] E. Grefenstette, B. Amos, D. Yarats, P. M. Htut, A. Molchanov, F. Meier, D. Kiela, K. Cho, and S. Chintala, *Generalized inner loop meta-learning*, *arXiv preprint arXiv:1910.01727* (2019).

[247] X. Zhu, Z. Ghahramani, and J. D. Lafferty, *Semi-supervised learning using gaussian fields and harmonic functions*, in *Proceedings of the 20th International Conference on Machine learning (ICML-03)*, pp. 912–919, 2003.

[248] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf, *Learning with local and global consistency*, in *Advances in Neural Information Processing Systems*, pp. 321–328, 2004.

[249] A. Jung, *On the duality between network flows and network lasso*, *IEEE Signal Processing Letters* (2020).

[250] W. Hamilton, Z. Ying, and J. Leskovec, *Inductive representation learning on large graphs*, in *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017.

[251] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, *Graph attention networks*, in *International Conference on Learning Representations*, 2018.

[252] H. Yao, X. Tang, H. Wei, G. Zheng, and Z. Li, *Revisiting spatial-temporal similarity: A deep learning framework for traffic prediction*, in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 5668–5675, 2019.

[253] C. F. Daganzo, *The cell transmission model: A dynamic representation of highway traffic consistent with the hydrodynamic theory*, *Transportation Research Part B: Methodological* **28** (1994), no. 4 269–287.

[254] C. F. Daganzo, *The cell transmission model, part ii: network traffic*, *Transportation Research Part B: Methodological* **29** (1995), no. 2 79–93.

[255] T. Mallick, P. Balaprakash, E. Rask, and J. Macfarlane, *Transfer learning with graph neural networks for short-term highway traffic forecasting*, in *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 10367–10374, IEEE, 2021.

[256] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, *Hierarchical graph representation learning with differentiable pooling*, in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 4805–4815, 2018.

[257] F. M. Bianchi, D. Grattarola, and C. Alippi, *Spectral clustering with graph neural networks for graph pooling*, in *International Conference on Machine Learning*, pp. 874–883, PMLR, 2020.

[258] C. Finn, P. Abbeel, and S. Levine, *Model-agnostic meta-learning for fast adaptation of deep networks*, in *International Conference on Machine Learning*, pp. 1126–1135, PMLR, 2017.

[259] A. Raghu, M. Raghu, S. Bengio, and O. Vinyals, *Rapid learning or feature reuse? towards understanding the effectiveness of maml*, in *International Conference on Learning Representations*, 2019.

[260] Z. Wen, Y. Fang, and Z. Liu, *Meta-inductive node classification across graphs*, in *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2021.