UNIVERSITY OF CALIFORNIA
SANTA CRUZ

TOWARDS FAST AND ACCURATE STRUCTURED PREDICTION

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

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE AND ENGINEERING

by

Sriram Srinivasan

September 2020

The Dissertation of Sriram Srinivasan
is approved:

Dr. Lise Getoor, Chair

Dr. Kristian Kersting

Dr. Rajarshi Guhaniyogi

Quentin Williams
Acting Vice Provost and Dean of Graduate Studies

Contents

List of Figures

List of Tables

Abstract

Towards Fast And Accurate Structured Prediction

by

Sriram Srinivasan

Complex tasks such as sequence labeling, collective classification, and activity recognition involve predicting outputs that are interdependent, a task known as *structured prediction*. Approaches such as structured support vector machines, conditional random fields, and statistical relational learning (SRL) frameworks are known to be effective at structured prediction. Of these approaches, SRL frameworks are unique as they combine ease of using logical statements with the power of probabilistic models. However, structured prediction using SRL frameworks face several challenges that affect both scalability and accuracy of these models. In this dissertation, I address four key challenges that significantly improves scalability and accuracy of SRL model at performing structured prediction task. First, *structured prediction using large graphical models*: graphical models generated through SRL frameworks for structured prediction tasks are often large, and this can make inference computationally expensive. Second, *memory-constrained structured prediction*: often performing structured prediction using large graphical models require a large amount of memory, which can be infeasible as some model sizes can grow to require terabytes of space. Third, *real-time structured prediction*: some applications require performing structured prediction tasks in real-time, which requires an extremely efficient inference engine that can perform model generation and inference in under a few milliseconds. Fourth, *the optimization of arbitrary user-defined evaluation function*: every application evaluates its structured prediction task via a unique evaluation function with arbitrary form, making it challenging to optimize them through well-known approaches such as gradient-descent. In this dissertation, I propose four general techniques that address these challenges and improve the scalability and accuracy of structured prediction tasks. First, I develop an approach to detect and exploit symmetries in large graphical models that make inference more tractable. Second, I develop a new framework that intertwines model generation and inference to perform structured prediction effectively. Further, this approach makes use of disk space and a smart in-memory cache to minimize the memory footprint and scale inference based on disk space rather than the main memory. Third, I derive a new inference procedure based on a second-order method, which reduces the inference time on small graphical models to under a millisecond enabling structured prediction to be performed online in real-time. Further, to demonstrate the effectiveness of this procedure, I introduce the concept of a *micrograph* to generate small effective graphical models and perform real-time structured prediction in the product search domain. Finally, I propose four parameter estimation approaches based on search strategies that can directly optimize any user-defined evaluation function by treating it as a black-box. These methods significantly improve the scalability and accuracy of structured prediction tasks and expand their scope to domains previously impossible.

A loving dedication to the one that is all and all that is one.

Acknowledgments

First and foremost, I would like to extend my heartfelt gratitude to my advisor Lise Getoor without whom this Ph.D. would have been impossible. Lise's wisdom, encouragement, and support shaped my research and led to many successful projects. I consider myself lucky to have gotten to work with such a great advisor. I joined the LINQS lab in the third year of my Ph. D, when my previous advisor had to leave due to unforeseen circumstances. Lise gladly accepted me under her and let me work on a project which quickly led to a successful publication, for which I am truly grateful. She has continuously supported me in every decision throughout my Ph. D. with my best interests at her heart. Her constant comments have pushed me to get better at every step and become a better researcher, presenter, and mentor. All the lessons I have received from her, I am sure, will help me overcome any obstacle I may face in my future. I sincerely thank her for believing in me, supporting me, and guiding me throughout my Ph. D.

Next, I would like to thank my previous advisor, Vishy, without whom I would have never started my Ph. D. at UCSC. Vishy has been a great mentor helping me and supporting me throughout, starting with my initial Amazon days. Since then, I have counted on him for both technical and personal advice. He has inspired me beyond technical research through his many activities, like bee-keeping and farming. I look forward to joining back with him at Amazon and working with him.

Next, I would like to thank my advancement committee members Kristian Kersting, Rajarshi Guhaniyogi, and David Helmbold, whose comments helped shape this dissertation. I would like to extend a special thanks to Kristian Kerting, who helped us out with his expertise in lifted inference and SRL, which helped shape my lifted inference and tandem inference paper.

This dissertation would not have been possible without the help of several of my co-authors. First, I would like to thank Golnoosh Farnadi, who was a postdoc with Lise when I joined. I got to work on several projects with her. Her hard work, insight, and comments helped many of our projects get published at top conferences. I sincerely thank her for all the effort and support. I would also like to thank Behrouz Babaki, with whom I worked on lifted inference, my first paper in my Ph. D. His work on the theorem in the paper was crucial for its success. A special and huge thanks to Eriq Augustine, who has an equal contribution in our tandem inference paper. The help Eriq provides in the lab goes much beyond co-authored papers. His presence fills the lab with energy, and many conversations with him have helped me understand several technical and non-technical topics in much better detail. Eriq's work on the PSL codebase has helped me in every single project, and his promptness in helping fix any challenge has been a boon to the LINQS lab. Next, I would like to thank Charles Dickens for being enthusiastic and prudent in helping me with the journal paper on weight learning. An extra thanks to him for sharing tips and helping me out in surfing waves. During my internship at Amazon, I got to work with Karthik Subbian, my manager, and Nikhil Rao, my mentor. I thank them for believing in me and letting me work on a problem of my choice, which, with their help, got published at CIKM and is now part of my dissertation.

During my Ph. D, I collaborated with several excellent researchers and co-authored several papers that did not appear in my dissertation. I would like to thank them for helping me expand my knowledge and understanding of the field. The work on DSMLR with Parameswaran Raman truly helped me understand large-scale systems and optimization. I got to work with Varun Embar on multiple challenging projects like structure learning. I consider myself lucky to have Varun and Parameswaran, my collaborators, also be my house/roommates. A heartfelt thanks to them for making my life during Ph. D. extremely pleasant. I thoroughly enjoyed every conversation at home and at work. Next, I would like to thank Ehsan Amid and Manfred Warmuth for letting me work with them on TTLR, which helped me understand robust optimization techniques. I would also like to thank Manfred for his advice on farming, health, and nutrition, which has genuinely inspired me to follow his path of self-sustenance. Next, I would like to thank Rajdipa Chowdhry for being a great mentee and working with me on fake news detection. Her dedicated work enabled us to submit a paper in a short period of three months. I would also like to thank Spencer Thompson and Pigi Kouki for introducing me to the “fair recommender system” and helping me understand the area of fairness in machine learning. Finally, thanks to my internship mentors Choon-hui Teo, Hsiang-fu Yu, and Leo Dirac at Amazon, who have helped expand my knowledge and understand different systems. A special thanks to Vijai Mohan, who mentored me for the past nine years, which has helped me substantially in both my professional and personal life. His help and advice made me excel in several projects at Amazon and helped me secure an excellent position.

Sincere thanks to my parents, Lakshmi and Srinivasan, who have made innumerable sacrifices in their life so I could be successful. This life would not be possible without them. A heartfelt thanks to my brother and sister-in-law, Krishnan and Rukhmini, for always supporting me in every aspect and being the best I could wish. Next, I would like to thank all my labmates, LINQS and ML lab , who felt more like family to me than labmates. Finally, I would like to thank all my friends whose constant support and encouragement always keeps me going.

Portions of this work were supported by NSF grants CCF-1740850 and IIS-1703331, AFRL, and DARPA. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/NBC, or the U.S. Government.

Chapter 1

Introduction

With the growth of areas such as online social networks, bioinformatics, dialog systems, and computer vision, the availability of multi-relational structured data has dramatically increased. Many of the prediction tasks in machine learning can be improved significantly by considering the data's structural information. Considering the structure of the problem leads to more complex inference tasks that involve predicting structured outputs. Traditional machine learning methods often assume that the data is independent and identically distributed (i.i.d) [20]. This assumption often misrepresents the real data as it ignores the structural information. The use of structure in data and structure in the outputs for prediction often requires joint predictions to output structured objects such as trees, graphs, and sequences. This general task is known as *structured prediction*. Over the years, many approaches, such as relational models, structured support vector machines, and conditional random fields, have been proposed to perform structured prediction [53, 152, 15, 66].

Of these approaches, statistical relational learning (SRL) frameworks [130, 55, 37, 36] are known to perform easy and effective structured prediction. SRL frameworks generate probabilistic graphical models that can perform structured predictions. SRL frameworks combine the convenience of representing the structure through logical relations and the power of probabilistic models at making accurate predictions. This makes SRL frameworks an attractive option for performing structured prediction. Models in an SRL framework are often defined using a set of weighted logical rules. These rules are instantiated/grounded with data to generate a probabilistic graphical model on which inference is performed. Models defined through these frameworks have been used in many fields and shown to yield state-of-the-art results. Examples include NLP tasks [16, 160], image processing [3, 59], bioinformatics [143], search [8], recommender systems [85, 90] and more [42, 43, 28].

Multiple SRL frameworks have been developed [130, 54, 11] to perform effective structured prediction. *Markov logic networks* (MLNs) [130] and *probabilistic soft logic* (PSL) [11] are two of the most widely used SRL frameworks that define probabilistic graphical models through weighted logical rules. A MLN program generates a discrete Markov random field (MRF) based on boolean logic called Markov networks, and PSL makes a continuous relaxation of the random variables using soft logic to generate a continuous MRF called a *hinge-loss Markov random field (HL-MRF)*. This relaxation allows the problem of MAP inference to take the form of a convex objective function. Empirical evidence shows that the ability to solve a convex optimization problem rather than a combinatorial problem can yield exponential speedups at inference time over other traditional quadratic programming based methods [11]. MLN and PSL have both shown to produce the state-of-the-art results in several domains [118, 46, 163, 159, 127, 86, 121, 144]. In this dissertation, I make use of these two SRL frameworks to address some of the key challenges in performing accurate and scalable structured prediction via SRL.

1.1 Structured Prediction Using SRL Frameworks

There are several challenges in SRL and graphical models which affect the scalability and accuracy of a model defined through MLNs and PSL. In this dissertation, I focus on addressing four significant challenges that improve the scalability and accuracy of these models. The first three challenges address three different scalability issues that exist in graphical models generated through SRL frameworks. The last challenge is related to the accuracy of the predictions made by SRL evaluated through arbitrary evaluation function. Below, I briefly introduce the four challenges tackled in this dissertation.

1.1.1 Challenge 1: Structured Prediction Using Large Graphical Models

The graphical models generated through SRL frameworks for structured prediction are often large. Performing inference on large graphical models is complex and inefficient. However, in many of these instances, the large graphical models contain repeated substructures or *symmetries*. Detecting these symmetries and eliminating them to produce a smaller equivalent problem can often lead to significant computational speedups for inference. The process of detecting and exploiting symmetries to perform faster inference is referred to as *lifted inference* [70, 77, 41, 68, 107, 119, 38, 71, 140, 41, 6, 106]. However, in SRL frameworks like PSL, for the lifting approach to be practical, the lifting process needs to be efficient and faster than performing full inference, which is already efficient.

1.1.2 Challenge 2: Memory-Constrained Structured Prediction

Graphical models used for the structured prediction can grow to extreme sizes. This means that the graphical models generated through SRL can quickly grow to sizes that cannot fit in memory. Consider a simple transitive rule in SRL: $Link(A, B) \wedge Link(B, C) \implies Link(A, C)$, commonly used in conjunction with link prediction. When instantiated with a small fully-connected network with 1000 entries, the graphical models generated will contain a billion cliques. This will require memory in the order of 100s of GB or TBs. While several different approaches such as lifted inference, approximate lifted inference [135, 141, 155, 23, 132, 24, 25, 72], hybrid approaches [9], distributed approaches [97], blocking [117], and other approaches [10] have been proposed, all of them fundamentally assume that the graphical model can fit in memory. This assumption severely restricts the application of SRL and graphical models to perform structured prediction.

1.1.3 Challenge 3: Real-time Structured Prediction

The first two challenges focus on inference in large models, but there is another class of problems where structured prediction needs to be performed at real-time with a stringent time constraint. These problems generally require us to generate and perform inference in under a few milliseconds. While there are previous works that focus on online inference using relational models [114, 154, 75, 123, 122], they do not assume any stringent time constraint and generally focus on continuous optimization rather than real-time predictions. To understand this challenge better, consider an information retrieval task on large e-commerce websites. A user enters a search query and expects to see a set of items that match the facets (or attributes) mentioned in the query. However, due to several lexical and behavioral patterns, the item set might have some facet mismatches. Performing collective classification [136] using graphical models to identify any mismatches between the items returned, and the query placed can significantly improve user experience on the e-commerce site [45]. However, creating a large graphical model with all items and queries and performing inference is infeasible as the online collective classification task (i.e., model generation and inference) needs to be completed under a few milliseconds. Any delay in this task leads to a significant negative impact on the user experience. For such setups, it is crucial to carefully generate small graphical models and perform extremely efficient inference for maximum impact. As mentioned earlier, many of the proposed approaches address the challenge of scaling to large graphical models, but performing inference on small graphical models with extreme time constraints has received little attention.

1.1.4 Challenge 4: Metric-Optimized Parameter Estimation

The fourth challenge relates to the accuracy of the structured prediction task using SRL frameworks. Learning the optimal set of weights is essential to making accurate predictions in SRL. Most approaches that attempt to find the optimal set of weights maximize the likelihood of the model [96, 139, 14]. This is a hard problem, as it involves the computation of an intractable log-partition function. Many approaches maximize approximations of the likelihood to perform efficient weight learning. However, a key challenge in using these methods is that the objective for most realworld problems is usually a user-defined evaluation metric, such as F1 score, and not the likelihood function. The inherent assumption made by likelihood-based methods that there is a correlation between likelihood and the user-defined evaluation function is not always correct. Therefore, most methods end up maximizing different versions of likelihood with no guarantees on improving the user-defined metric. While it is paramount for us to learn the optimal set of weights that maximize a user-defined evaluation function, it is extremely challenging due to the large set of possible evaluation functions. Generating a method for every evaluation function is infeasible and some evaluation functions cannot be expressed in closed-form function, precluding closed-form updates.

1.2 Contributions

In this dissertation, I introduce several strategies to address the four challenges mentioned above. For the first three challenges I introduce three orthogonal approaches that directly improve scalability by: 1) efficiently identifying and exploiting symmetries in graphical models; 2) smart usage of disk space with main memory to constrain the memory footprint; and 3) introducing the concept of micrographs and using second-order methods to perform inference efficiently. Finally, as the solution to the last challenge, I introduce four novel parameter learning techniques. While I use PSL as the main SRL framework for the first three challenges and both PSL and MLN for the fourth, the strategies introduced are general and can be extended and used in other SRL frameworks and graphical models. Below, I highlight the key contributions provided in this dissertation. Portions of the solutions proposed in this dissertation have been published in [148, 149, 150, 147].

To address the first challenge of scaling inference to large graphical models, I introduce a new approach to perform lifted inference in HL-MRFs referred to as *lifted HL-MRFs* (LHL-MRFs). HL-MRFs make use of an efficient optimizers, alternating directions method of multipliers (ADMM) [21], to perform efficient large-scale inference (explained in Section 2.3). However, HL-MRFs generated through PSL might have symmetries that could increase the computational cost of inference by performing redundant operations. The color refinement algorithm (explained in Section 3.3.1) has been shown to be efficient at identifying symmetries in linear and quadratic programs [104, 108]. In this approach, I combine the color refinement algorithm with ADMM to perform efficient lifted inference in HL-MRFs. I show that the lifted network generated by the color refinement algorithm for any HL-MRF is correct and conforms to a form amenable to ADMM. I also show that performing inference on LHL-MRFs is equivalent to performing inference on HL-MRFs. The quasilinear time complexity of our lifting approach and the ability to use ADMM to perform inference ensures a significant runtime reduction in the inference process. I perform an empirical evaluation of our approach on three realworld datasets, and show that even on datasets with $\sim 20\%$ reduction in the model size, we gain $\sim 10\%$ in runtime performance with no loss of accuracy. Further, using synthetic datasets, I perform a thorough study to understand the impact of lifting on HL-MRFs with varied graph densities, precision in data, and symmetries. I show that the liftability of a HL-MRF is inversely correlated to the data precision, i.e., high precision in data reduces the liftability as high precision in data can be susceptible to noise breaking symmetries. I also show that the runtime gains and the amount of symmetry in the HL-MRF are directly correlated.

Next, I address the second challenge by introducing a novel approach referred to as *tandem inference* (TI) that is orthogonal to the lifted inference method proposed earlier. TI works by intertwining the process of model generation (grounding) and inference to make them run in tandem. Further, it uses smart caching and disk space to minimize the memory footprint in the RAM, making it possible to perform inference in prohibitively large models on machines with low RAM. I introduce two new strategies that enable the possibility of TI: 1) continuously generate cliques of the graphical model rather than a full instantiation; and 2) a gradient-based optimization that can perform inference using one clique at a time. Further, I stream cliques to and from a disk and instantiate a limited size cache to hold cliques in memory for inference. I perform a comprehensive empirical evaluation of TI and compare it with traditional inference on eight realworld datasets and two synthetic datasets. In the experiments, I show that TI can perform inference on extremely large models, which contain a billion cliques, on a machine with just 10GB of RAM, where a traditional inference would require over 800GB of RAM. Further, these experiments also show that TI can even run up to 8 times faster than traditional approaches on our largest realworld dataset. I also perform a varied set of experiments to show the impact of different cache sizes on inference and the convergence of different the new gradient-based approach and traditional approach.

The next contribution of this dissertation involves addressing time-constrained inference in graphical models for real-time structured prediction. To do this, I choose a specific application of product search/retrieval and a collective classification task. First, I introduce a new task in product retrieval referred to as *facet mismatch classification* (FMC) and then propose an approach that is both accurate and efficient to be performed at runtime with a stringent time constraint. Every product on an e-commerce website is associated with a set of attributes called facets. A user generally enters a search query and attempts to match the query with the facets of a product of interest. For example, a user enters a search query “iPhone 8 64GB” and search engine needs to match products along all facets “iPhone 8” and “64GB”. When a product returned does not match along any of the facets, we refer to it as a facet mismatch (FM). Given a query and a set of products, the general task of FMC is to classify all products in this set as a FM or not. While traditional methods assume independence between products, I introduce a structured mismatch classification (SMC) approach that considers the similarities between products at the time of classification. To efficiently perform SMC, I introduce the concept of a *micrograph* that generates small graphical models by conditioning on a query and considering only the links between products such as the textual similarity between product titles. To perform efficient inference on these graphical models generated, I propose an approach based trust region Newton method (TRON) [93]. The TRON-based inference has a high convergence rate, making them practical for real-time systems. I also show that inference in PSL can be cast as learning a one-class support vector machine (SVM) problem. This enables PSL to use any optimization approach that is available for SVMs. In the empirical evaluation, I show that perform SMC can yield up to 12% improvement in precision across three different e-commerce datasets. Further, we compare the runtimes between TRON and ADMM for SMC and observe that the proposed TRON-based approach is 150x faster than the ADMM-based method and the classification on a micrograph takes under 1ms which makes it feasible to perform this inference in real-time.

The contributions mentioned so far are orthogonal approaches that address the challenges in the scalability of SRL-based systems (in specific PSL). While it was possible to use the existing ADMM-based inference in LHL-MRF effectively, I introduced a stochastic gradient-based approach to perform TI and a TRON-based approach to perform real-time inference using PSL. These three optimizers complement each other and help improve the overall scalability of PSL and its applicability to different applications. While the ADMM-based inference for PSL is efficient and embarrassingly parallel, it does not easily conform to streaming architecture and uses excess memory by creating many auxiliary variables. However, a stochastic gradient-based approach easily conforms to a streaming architecture and does not use excess memory, but is not embarrassingly parallel and is more sensitive to hyperparameters than ADMM. While both these inference methods have a low per-iteration cost, they also have a low convergence rate and take many iterations to converge. However, a TRON-based approach has a high convergence rate and takes fewer iterations to converge, but its per-iteration cost increases with model size, making them less practical for large models. Therefore, these three approaches can be effective in different scenarios and help improve the overall performance of PSL. For instance, while PSL with ADMM is ideal in a distributed setting, PSL with a stochastic gradient-based approach is suitable for an online/streaming setting, and PSL with TRON is perfect in applications requiring real-time inference using small graphical models. Hence, the combination of the contributions so far significantly improves the scalability and applicability of PSL.

This dissertation's final contribution addresses the fourth challenge of performing parameter estimation to maximize any user-defined evaluation metric. To do this, I introduce four novel weight learning approaches for SRL frameworks based on search strategies commonly used for hyperparameter tuning in statistical machine learning. These approaches are based on black-box optimization and can optimize for any arbitrary evaluation function without the need to re-derive update equations. The first two approaches are based on "random search", which has been shown to be surprisingly effective at searching for hyperparameters in deep learning [17]. The third approach is based on Hyperband [91], which performs a smart search through effective resource allocation. The fourth approach is based on Bayesian optimization with Gaussian process regression, which is an efficient and robust approach for approximating arbitrary functions. I showcase the power of these approaches by applying them to both MLNs and PSL. To ensure an efficient searching of the weight space using these approaches, I introduce a new projection, which I refer to as *scaled space* (SS), that is accurate at representing the weights. I show that SS eliminates the redundancies in the original weight space and ensures that the distances in this space are correlated with the true representation of the weights. Because sampling weights from SS can be challenging, I introduce an approximation of SS and an approach to sample efficiently form this approximation. I perform an elaborate set of empirical evaluations on five realworld datasets with two metrics each to compare the performance of 8 different weight learning techniques in MLNs and PSL. I show that the search-based approach introduced almost always outperforms the previous methods by up to 10% in different evaluation metrics. Further, runtimes measured for these approaches show that the search-based methods are scalable with the size of the graphical model generated. Finally, I show that all the four search-based approaches are both accurate and robust to weight initializations, and that the Bayesian optimization approach is also robust to other hyperparameters.

1.3 Organization

This dissertation is organized as follows. First, in Chapter 2, I provide the background necessary to understand logical MRFs, HL-MRFs, and PSL. Next, in Chapter 3, I elaborate my approach, LHL-MRF, to identify and exploit symmetry to perform efficient inference in HL-MRFs. Then, in Chapter 4, I explain TI to perform efficient memory-constrained inference in graphical models generated from SRL frameworks. Further, in Chapter 5, I propose a TRON-based inference approach for real-time structured prediction in PSL and show its effectiveness through a novel application in the domain of product retrieval. Next, in Chapter 6, I describe four new weight learning approaches for SRL. Finally, in Chapter 7, I provide the conclusion and potential future directions of this research.

Chapter 2

Background in SRL

In this chapter, I review the basics of SRL approaches that use weighted logical rules to define a probabilistic graphical model. In specific, I give an overview of two powerful SRL frameworks MLNs and PSL. I highlight their similarities and differences and discuss the inference and weight learning techniques commonly used with them.

2.1 Statistical Relational Learning

SRL methods such as PSL and MLNs combine the power of probabilistic graphical models with weighted first-order logical rules to capture relational structure in any domain. A set of weighted first-order logical rules are instantiated with data D to generate a logical Markov random field (LMRF). This process is referred to as grounding and every instantiated rule consisting of only ground predicates is called a ground rule. Every ground predicate represents a random variable in the LMRF which may be observed (x) or unobserved (y) and a ground rule represents a clique potential ϕ in the LMRF. Every potential in the LMRF is associated with a weight equal to the weight assigned to the logical rule it was instantiated from. The weight of a logical rule represents the importance of the rule in the model. The weighted sum of potentials in the LMRF is referred to as the energy function $\mathbf{E}$. A LMRF can be formally defined as:

Definition 1 (Logical Markov random fields)**.** *Let* $\mathbf{y} = \{y_1, y_2, ..., y_n\}$ *be* n *unobserved random variables,* $\mathbf{x} = \{x_1, x_2, ..., x_m\}$ *be* m *observed variables or evidence, and* $\phi = \{\phi_1, \phi_2, ..., \phi_\iota\}$ *be* ι *potentials describing different logical relations between variables. The output of a potential function* $\phi_i(\mathbf{x}, \mathbf{y})$ *is a real-valued scalar representing compliance of* $\mathbf{x}$ *and* $\mathbf{y}$ *with* ϕ_i*. Further, let* $\mathbf{w} \in \{w_1, w_2, \dots, w_\iota\}$ *be a set of weights associated with each potential. The energy function of a LMRF is defined as:*

$$\mathbf{E}(\mathbf{y}|\mathbf{x}) = \sum_{i=1}^{\iota} w_i \phi_i(\mathbf{x}, \mathbf{y}) \qquad \textit{s.t., } \mathbf{y} \in \{0,1\}^n;\ \mathbf{x} \in \{0,1\}^m \tag{2.1}$$

and the conditional likelihood of a LMRF is defined as:

$$P(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \frac{1}{Z(\mathbf{y})} exp(\hat{s} \cdot \mathbf{E}(\mathbf{y}|\mathbf{x})) \tag{2.2}$$

where $Z(\mathbf{y}) = \int_{\mathbf{y}} exp(\hat{s} \cdot E(\mathbf{y}|\mathbf{x}))$ *is a normalization constant and* $\hat{s} \in \{1, -1\}$ *determines the sign in the exponent based on if potential measures satisfaction or dissatisfaction.*

To better understand LMRFs and the process of grounding, consider a simple collective labeling problem:

Example 1. *Assume we have a set of users* $\mathbf{U}$ *and a label that can be either* $true$ *or* $false$ *associated with each user, such as if a user* $Smokes$ *or not. The label for* $Smokes$ *is observed for some users (*$\mathbf{U}_o$*) and unobserved for the rest (*$\mathbf{U}_u$*). The task is to infer the labels for* $\mathbf{U}_u$*. Let the input data include a social network with friendship links between users* $\mathbf{U}$*,* $Friend(U, V)$*, and a local predictor that predicts if a user smokes or not based on the user's features,* $LocalPredictor(U)$*. The* $LocalPredictor$ *is a classifier (e.g., logistic regression, neural network, decision tree) built based on* *attributes of the individual U, for example $SmellSmoky(U)$ and $YellowFingers(U)$. Below is a simple SRL model for collectively inferring labels:*

$$w_1 : \textit{LocalPredictor}(U) \rightarrow \textit{Smokes}(U)$$

$$w_2 : \textit{Smokes}(U) \wedge \textit{Friend}(U,V) \rightarrow \textit{Smokes}(V)$$

where w_1 and w_2 are weights for the rules. The above model is then grounded with users $\mathbf{U}$ to generate an LMRF. Each ground rule (e.g., w_1: LocalPredictor(“Bob”) → Smokes(“Bob”)) generates a potential function ϕ_i. Each ground predicate created by instantiating the $Smokes$ predicate with users $\mathbf{U}_u$ generates a set of unobserved random variables $\mathbf{y}$ and the rest of the ground predicates generate a set of observed random variables $\mathbf{x}$. Fig. 2.1 shows the resulting graphical model when instantiated over a small social network of 100 individuals.

Inference in a LMRF is performed by finding a maximum aposteriori estimate (MAP) of the random variables $\mathbf{y}$ given evidence $\mathbf{x}$. This is performed by maximizing the density function or equivalently maximizing the energy function in Equation 2.1. MAP inference is expressed as:

$$\underset{\mathbf{y}}{\operatorname{argmax}}\, P(\mathbf{y}|\mathbf{x}) = \underset{\mathbf{y}}{\operatorname{argmax}}\, \hat{s} \cdot \mathbf{E}(\mathbf{y}|\mathbf{x}) \tag{2.3}$$

MLN and PSL make specific choices in LMRF to generate a Markov network and a hinge-loss Markov random field (HL-MRF), respectively. Next, I highlight the choices made by MLNs, followed by PSL, along with weight learning approaches commonly used by both.

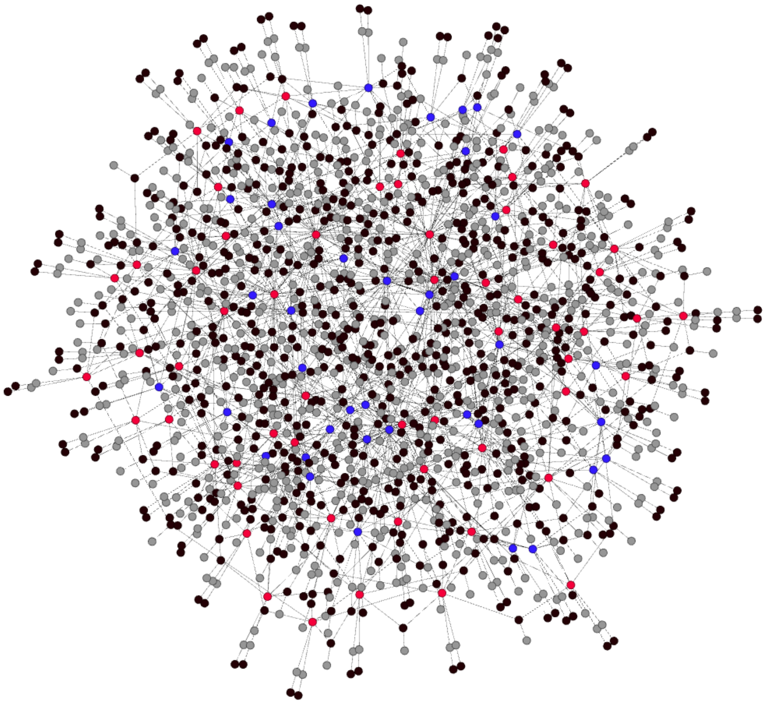


Figure 2.1: Factor graph produced by grounding the example SRL model with synthetic data for 100 users. The blue nodes are users who smoke, and the red nodes are users who do not smoke. Grey nodes are the rest of the grounded atoms, and the black nodes are potentials. Here, we see that the resulting factor graph is large, complex, and highly connected even for this simple case.

2.2 Markov Logic Networks

MLNs [130] use boolean logic and use discrete random variables as in LMRF. Potential functions ϕ are indicator functions that are one if a ground rule is satisfied and zero otherwise. Every ground logical clause can be written as:

$$n_i(\mathbf{x}, \mathbf{y}) = \Big(\bigvee_{i \in I_u^+} y_i \Big) \vee \Big(\bigvee_{j \in I_u^-} \neg y_j \vee \bigvee_{k \in I_o^+} x_k \Big) \vee \Big(\bigvee_{l \in I_o^-} \neg x_l \Big) \tag{2.4}$$

where I_u^+ and I_o^+ are sets of unobserved and observed positive literals that participate in the clause and I_u^- and I_o^- are sets of unobserved and observed literals that participate in the cause with a negation. A potential in MLN is of the form:

$$\phi_i(\mathbf{x}, \mathbf{y}) = n_i(\mathbf{x}, \mathbf{y}) = \min \left\{ \sum_{i \in I_u^+} y_i + \sum_{j \in I_u^-} (1 - y_j) + \sum_{k \in I_o^+} x_k + \sum_{l \in I_o^-} (1 - x_l), 1 \right\} \tag{2.5}$$

where $n_i(\cdot, \cdot)$ is the satisfaction of the i^{th} ground rule. Since the potentials measure the satisfaction of the ground rules, the $\hat{s}$ is set to one and the MAP inference in MLN is written as:

$$\operatorname*{argmax}_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}) = \operatorname*{argmax}_{\mathbf{y}} \mathbf{E}(\mathbf{y}|\mathbf{x}) \tag{2.6}$$

Many different efficient implementations of MLNs exist [138, 116, 158, 65], but in this dissertation I choose Tuffy [116] as our MLN framework. Tuffy combines a hybrid architecture with smart data partitioning to scale up inference in MLN. Tuffy uses the WalkSAT algorithm [67, 116] to perform MAP inference in MLNs. WalkSAT is an approximate search-based approach that iteratively flips random variables' values to find the most satisfied world. Every step, WalkSAT chooses an unsatisfied ground rule and flips a variable at random or flips a variable that would make the ground rule most satisfied. After a set amount of iterations, WalkSAT returns the best assignments for the random variables obtained.

Many weight learning approaches have been proposed for MLNs [130, 139, 96]. It has been observed that second-order methods for weight learning tend to perform better than other approaches [96]. Tuffy uses a second-order method, the diagonal Newton approach introduced in [96], to perform weight learning.

2.2.1 Diagonal Newton Method for Weight Learning (DN)

This approach minimizes the negative conditional log-likelihood (CLL) using a Newton-based method. A Newton-based approach reaches global minimum by multiplying the gradient with the inverse of the Hessian at every iteration:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{H}^{-1}\mathbf{g}$$

where $\mathbf{H}$ is the Hessian, $\mathbf{g}$ is the gradient w.r.t. $\mathbf{w}$, and t is the iteration number. Since computing the Hessian can be expensive and infeasible, a diagonal Newton method is used as an approximation for the Hessian. The Hessian of the negative CLL is the covariance matrix of the CLL, and this is approximated through samples generated using MC-SAT [120]. The final update for the weight of a logical rule at each iteration is given by:

$$w_i = w_i - \alpha \frac{\mathbb{E}_{\mathbf{w}}(N_i) - N_i}{\mathbb{E}_{\mathbf{w}}(N_i^2) - (\mathbb{E}_{\mathbf{w}}(N_i))^2} \tag{2.7}$$

where $N_i = \sum_{j \in g_i} n_j(\mathbf{x}, \mathbf{y})$, g_i is a set of ground rules generated by the i^{th} rule, $\mathbb{E}_{\mathbf{w}}$ is expectation w.r.t. the weights and α is the step size.

2.3 Probabilistic Soft Logic

PSL [11], unlike MLNs, uses soft logic and relaxes random variables to be in the range $[0, 1]$. Specifically, PSL uses Łukasiewicz logic to generate potentials which take the form of hinges. In Łukasiewicz logic conjunction ($\tilde{\wedge}$), disjunction ($\tilde{\vee}$) and negation ($\tilde{\neg}$) are defined as:

$$y_1 \tilde{\wedge} y_2 = \max\{y_1 + y_2 - 1, 0\} \tag{2.8}$$

$$y_1 \tilde{\vee} y_2 = \min\{y_1 + y_2, 1\} \tag{2.9}$$

$$\tilde{\neg} y = 1 - y \tag{2.10}$$

The ˜ indicates the relaxation over Boolean values. These logical statements are flipped to measure the distance to satisfaction of a ground rule instead of their satisfaction and are used as potential function ϕ. A hinge potential in PSL is of the form:

$$\phi_i(\mathbf{x}, \mathbf{y}) = \max(\ell_i(\mathbf{x}, \mathbf{y}), 0)^{d_i} \qquad \textit{s.t.,}\ \mathbf{0} \leq \mathbf{y} \leq \mathbf{1}\ ;\ \mathbf{0} \leq \mathbf{x} \leq \mathbf{1} \tag{2.11}$$

$$\ell_i(\mathbf{x}, \mathbf{y}) = 1 - \sum_{i \in I_u^+} y_i - \sum_{j \in I_u^-} (1 - y_j) - \sum_{k \in I_o^+} x_k - \sum_{l \in I_o^-} (1 - x_l) = y^T z_i - c_i \tag{2.12}$$

where ℓ_i is a linear function and $d_i \in \{1, 2\}$ provides a choice of two different loss functions, $d_i = 1$ (i.e., linear) and $d_i = 2$ (i.e, quadratic), c_i is the constant associated with the potential, and $z_i \in \{0, 1, -1\}^n$ is a vector that determines how all the variables participate in a specific potential i. $z_{i,j} = 0$ implies variable y_j does not participate, $z_{i,j} = 1$ implies variable $y_j \in I_u^-$, and $z_{i,j} = -1$ implies variable $y_j \in I_u^+$. The constant c_i is computed based on the observed variables xs and other constants in the equation. Since the potential functions in PSL measure distance to satisfaction, the $\hat{s}$ is set to -1. Further, weights in PSL are positive and real, i.e., $w_i \in \mathbb{R}^+$. The MAP inference in PSL is expressed as:

$$\underset{\mathbf{y}}{\operatorname{argmax}} P(\mathbf{y}|\mathbf{x}) = \underset{\mathbf{y}}{\operatorname{argmin}} \mathbf{E}(\mathbf{y}|\mathbf{x}) \tag{2.13}$$

A key advantage of using PSL is that the inference objective is convex. This enables the use of efficient convex optimization procedures, such as the alternating direction method of multipliers (ADMM) [21]. Hence, given known weights, inference in PSL can be performed at scale, enabling predictions on large realworld datasets.

The first step in solving the problem with ADMM is to form the *augmented Lagrangian function* of the problem as (ignoring $\boldsymbol{x}$ in the equations as they are observed):

$$\begin{aligned} L(y, y_l) = \min_{y, y_l} \sum_{i=1}^{\iota} w_i \phi_i(y_{l,i}) + \sum_{j=1}^{n} \mathcal{X}_{[0,1]}[y_j] \\ \textit{subject to: } y_{l,i} = y \; \forall i \in \{1, ..., m\} \end{aligned} \tag{2.14}$$

where $\mathcal{X}_{[0,1]}[y_j]$ is an indicator function which produces zero if $y_i \in [0, 1]$ and infinity otherwise, and y_l is a matrix with m rows and n columns and $y_{l,i}$ represents i^{th} row of the matrix. The augmented Lagrangian form of this would be:

$$\begin{aligned} L(y, y_l, \alpha) = \min_{y, y_l} \sum_{i=1}^{\iota} w_i \phi_i(y_{l,i}) + \sum_{j=1}^{n} \mathcal{X}_{[0,1]}[y_j] \\ + \sum_{i=1}^{\iota} \alpha_i^T (y_{l,i} - y) + \frac{\rho}{2} \sum_{i=1}^{\iota} ||y_{l,i} - y||_2^2 \end{aligned}$$

where $\rho > 0$ is step size and α is a matrix of same dimension as y_l and represents the dual variables. The update equation for ADMM at iteration t is the following:

$$\alpha_i^t = \alpha_i^{t-1} + \rho(y_{l,i}^{t-1} - y^{t-1})$$
$$y_l^t = \underset{y_l}{\operatorname{argmin}}\, L(y_l, \alpha^t, y^{t-1})$$
$$y^t = \underset{y}{\operatorname{argmin}}\, L((y_l, \alpha^t, y^{t-1})$$

The ADMM updates ensure that y converges to the MAP state.

Unlike MAP inference, the task of learning the rule weights from training data is not as efficient (although, as we will see in Chapter 6, having tractable MAP inference is useful for weight learning). There are three primary approaches used to perform weight learning in PSL as discussed in [14]: Maximum Likelihood Estimation (MLE), Maximum Pseudolikelihood Estimation (MPLE), and Large-Margin Estimation (LME).

2.3.1 Maximum Likelihood Estimation (MLE)

This approach maximizes the log-likelihood function with respect to the weights of the rules based on the training data. Since all the potentials generated by a rule share the same weight, Equation 2.1 can be written as $\sum_{i=1}^{r}[w_i\Phi_i(\mathbf{x},\mathbf{y})]$ where r is the number of template rules, w_i represents the weight of the i^{th} rule, $\Phi_i = \sum_{j\in g_i}\phi_j(\mathbf{x},\mathbf{y})$, and g_i is a set of ground rules generated by the i^{th} rule. The partial derivative of the log of the likelihood function given in Equation 2.2 for PSL with respect to w_q, for $q \in \{1,\ldots,r\}$ is:

$$\frac{\partial logP(\mathbf{y}|\mathbf{x})}{\partial w_q} = \mathbb{E}_{\mathbf{w}}\Big[\Phi_q(\mathbf{x},\mathbf{y})\Big] - \Phi_q(\mathbf{x},\mathbf{y}) \qquad (2.15)$$

where $\mathbf{w} = \{w_1,...,w_r\}$ and $\mathbb{E}_{\mathbf{w}}$ is expectation w.r.t. the weights. It is infeasible to compute the expectation, hence to make the learning tractable, a MAP approximation is used that replaces the expectation in the gradient with the corresponding values in the MAP state. This approach is a structured variant of the voted perceptron algorithm [32].

2.3.2 Maximum Pseudolikelihood Estimation (MPLE)

An alternative approach that maximizes the pseudolikelihood function, which is given by:

$$P^*(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{n} P^*(y_i|MB(y_i), \mathbf{x}) \tag{2.16}$$

where n is the number of random variables and $MB(y_i)$ is the Markov blanket of y_i. Equation 2.16 is maximized using a gradient ascent based approach and the derivative of the log-pseudolikelihood function with respect to w_q is given by:

$$\frac{\partial logP^*(\mathbf{y}|\mathbf{x})}{\partial w_q} = \sum_{i=1}^{n} \mathbb{E}_{y_i|MB}\Big[\sum_{j \in g_q : i \in \phi_j} \phi_j(\mathbf{x}, \mathbf{y})\Big] - \Phi_q(\mathbf{x}, \mathbf{y}) \tag{2.17}$$

where $i \in \phi_j$ implies that variable y_i participates in the potential ϕ_j. Using a Monte Carlo approach this derivative can be computed in linear time in the size of $\mathbf{y}$.

2.3.3 Large-Margin Estimation (LME)

This approach focuses on maximizing the MAP state rather than producing accurate probabilistic models. This approach uses the intuition that the ground-truth state $\mathbf{y}$ should have energy lower than any alternate state $\tilde{\mathbf{y}}$ by a large margin defined by a loss function L. The objective function to find the optimal set of weights is given by:

$$\begin{aligned} \mathbf{w}^* = \operatorname*{argmin}_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 + C\xi \\ s.t: \mathbf{w}^T(\Phi(\tilde{\mathbf{y}}, \mathbf{x}) - \Phi(\mathbf{y}, \mathbf{x})) \leq -L(\mathbf{y}, \tilde{\mathbf{y}}) + \xi \end{aligned} \tag{2.18}$$

where L is a loss function such as the L1 distance between $\mathbf{y}$ and $\tilde{\mathbf{y}}$, and ξ is a slack variable. Equation 2.18 is then solved by performing a large-margin estimation based on a cutting-plane approach for structural support vector machines [66].

Chapter 3

Lifted hinge-loss Markov random field

In this chapter[1] I discuss in detail about the existence of symmetry in the graphical model generated through PSL. Further, I describe an approach that can be used to find symmetries in these model and use them to improve the runtime of the inference task. Through evaluation on realworld datasets I show the effectiveness of our approach in performing lifted inference in PSL.

3.1 Introduction

For SRL frameworks, exact inference is often computationally expensive, because inference is performed over large grounded graphical models. However, this ground representation is typically derived from a much smaller set of logical rules, and, depending on the data, it often contains identical substructures. These identical substructures cause unnecessary work for the inference algorithm by repeatedly performing the same operations. *Lifted inference* [70, 77, 41, 68] aims to detect common substructures and use them to avoid redundant computations. There have been many approaches introduced to perform lifted inference which we mention in our related work section.

[1] Appeared in *33rd AAAI Conference on Artificial Intelligence (2019)*

The inference algorithm in HL-MRFs relies on *alternating direction method of multipliers (ADMM)*. ADMM is an iterative optimization method [21] that provides an elegant approach for finding the saddle point in augmented Lagrangian. The ADMM algorithm for HL-MRFs use the structure in the objective function and solves the subproblems in each iteration using closed-form solutions.

Our work integrates the concept of lifting using color refinement algorithm with ADMM to perform a more efficient inference in HL-MRFs. Using ADMM for HL-MRFs [11] showed exponential performance gains over traditional LP/QP solvers. To our best knowledge this is the first approach that combines ADMM with color refinement to perform lifting for probabilistic inference. Our contributions are as follows: 1) we propose the first method for detecting and eliminating the symmetries in HL-MRFs inference problems using color refienment algorithm. By applying this method to the realworld datasets, we observe significant reductions (up to 66%) in the size of problems, 2) we show how the lifted problem can be cast back into the same form as the original inference problem, and solved using the specialized inference algorithm of HL-MRFs. The proposed integration of lifted inference and ADMM is essential to our goal. We solve the lifted problem using exsiting off-the-shelf solvers and the ADMM method, and demonstrate that lifting has better pay off when latter is employed, 3) we run a series of experiments on synthetically generated data, analyze the complicated relationships graph structures have with lifting, and show the effectiveness of LHL-MRFs on varied levels of symmetry.

3.2 Related work

Lifted inference in SRL is a well-studied problem. A popular approach is to group objects that are indistinguishable given evidence, and perform inference by operating on these groups. First-order variable elimination [119, 38] extends the standard variable elimination algorithm by summing over entire groups of random variables instead of one at a time. Lifted belief propagation [140, 72, 7, 6] employs the same message-passing method as the standard belief propagation algorithm. It first groups variables and form super nodes which are connected via so-called super edges. Message passing is then performed over the graph with these super nodes and super edges. A modified version of belief propagation(BP) called counting BP [71] constructs a compressed factor graph by creating clusternodes and clusterfactors and using a modified BP to perform inference. Some inference algorithms use the logical structure in a model for problem decomposition [39, 27]. The lifted versions of these algorithms perform this decomposition at the first-order level [56, 41].

The *exact* lifting methods discussed above assume that variables in the problem of interest are discrete. This makes them inapplicable to languages such as PSL, which are defined over continuous random variables. Recently developed lifted linear programming [104] and lifted convex quadratic programming [108] offer a method for finding and exploiting symmetries in linear programming and quadratic problems. Lifted linear and quadratic programming, groups indistinguishable variables using color refinement algorithm to produce a smaller linear or quadratic program making inference faster.

3.3 Background

In this section we discuss the color refinement algorithm on which our method is based on.

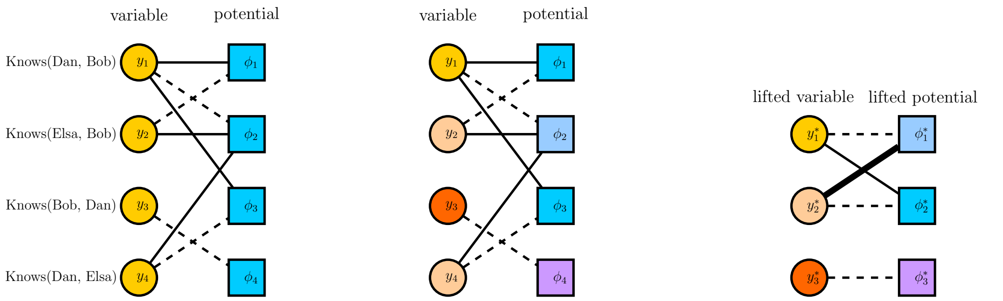
3.3.1 Color refinement

Color refinement is a simple algorithm to identify similar nodes in a graph [125]. This algorithm has efficient implementations that run in quasilinear time [31] and has been already used in practical graph isomorphism tools and for lifted inference [60]. The color refinement is an iterative algorithm that assigns colors to nodes in a sequence of refinement rounds. For a graph $G = (V, E)$, it first initializes all nodes in V with the same color. In every refinement round, any two nodes $v, w \in V$ with the same color are re-assigned to a different colors if there is some color c such that v and w have a different number of neighbors with color c; otherwise the no change is made. The refinement stops when the color of all pairs of nodes before and after the refinement round remains the same. This state of the graph where the colors of nodes do not change across refinement rounds is called a *stable coloring* of the graph. Let A be the adjacency matrix of G. Then the nodes u and v have the same color in the stable coloring of G iff it holds for every color C that $\sum_{w \in C} A_{vw} = \sum_{w \in C} A_{v'w}$.

The color refinement algorithm can be generalized to weighted graphs by refining the algorithm based on weighted sum of the edges (i.e., weighted sum of the edges of neighbors with the same colors) instead of degree (i.e., number of neighbors with the same colors). This generalization can then be extended to weighted bipartite graphs. However, the initial coloring in a bipartite graph is different. Instead of starting with one color, two initial color classes c_1 and c_2 are used where each color is assigned to one type of node. For instance in a factor graph, factors are initialized with color c_1 and nodes are initialized with color c_2. The final state of a bipartite graphs once the refinement stops is called a *stable bi-coloring* of the graph. At this state, the condition mentioned above holds for the weighted adjacency matrix of G.

3.4 Method

Graphical models generated from logical templates can manifest degrees of symmetry. In this section we introduce a method based on the color refinement algorithm to find and eliminate such symmetries in an HL-MRF energy function. The function



(a) Factor graph: initial coloring **(b)** Colored factor graph **(c)** Lifted factor graph

Figure 3.1: The factor graph of the HL-MRF model presented in Example 1. The labels of the factor nodes appear on their right side. Edge weights are represented by line style (solid: 1, dashed: -1, thick: 2).

obtained by this method is also a HL-MRF energy function. Preserving this form is crucial, since the efficient ADMM method introduced earlier is tailored for functions of this form. At the end of this section, we show that we can obtain the solution of the original problem by solving the lifted problem and mapping its solution back to the original space.

To understand our approach better we will introduce a simple PSL model and generate a small HL-MRF. Throughout this chapter we will use this as our running example:

Example 2. *Consider a PSL program consisting of a single rule that represents a transitivity relation among people:*

$$w : Knows(P1, P2) \wedge Knows(P2, P3) \rightarrow Knows(P1, P3) \wedge 2$$

where w is the weight of the rule. To keep the example simple we have only one squared rule. We have three individuals in our data: Bob, Dan *and* Elsa. *Given the observations $I\Big(\text{Knows(Ben,Elsa)}\Big) = 1$, $I\Big(\text{Knows(Elsa,Dan)}\Big) = 1$, and assuming that $I\Big(\text{Knows(X,X)}\Big) = 0$ for every individual X, our aim is to infer truth values for the* *remaining atoms. The grounded model consists of four atoms that participate in four grounded rules. Let us denote the unknown truth values by variables $y_1 \dots y_4$ (see Fig. 3.1a). The objective function for inference will be:*

$$
\begin{aligned}
\mathbf{E}(\boldsymbol{y}) &= w \max(2y_1 - y_2, 0)^2 + w \max(-y_1 + y_2 + y_4 - 1, 0)^2 \\
&\quad + w \max(y_1 - y_4, 0)^2 + w \max(-y_3 + 1, 0)^2 \qquad (3.1)
\end{aligned}
$$

3.4.1 Lifted HL-MRFs (LHL-MRFs)

Our lifting method operates on a *factor graph*, which is a graphical representation of a HL-MRF energy function.

Definition 2 (Factor Graph)**.** *The factor graph of an HL-MRF energy function is a graph $G = (U, V, E)$ in which there is a node $u_j \in U$ for each variable y_j ($j = 1, \dots, n$) and a node $v_i \in V$ for each potential ϕ_i ($i = 1, \dots, \iota$). For each nonzero coefficient z_{ij} of variable y_j in potential ϕ_i there is an edge $e_{ij} \in E$ between u_j and v_i with the weight z_{ij}. Each node $v_i \in V$ is labeled by the tuple (w_i, c_i, d_i).*

Example 3. *The energy function of the HL-MRF in Example 2 can be represented by the factor graph in Fig. 3.1a.*

We will now describe a method that given the factor graph G of an energy function $\mathbf{E}$, produces a potentially smaller factor graph G'. Instead of solving the MAP inference problem for $\mathbf{E}$, one can solve the MAP inference problem for the function $\mathbf{E}'$ represented by G' and map the solution back to the variables in $\mathbf{E}$.

We first assign the initial colors to the nodes of $G = (U, V, E)$. The nodes in V receive different colors based on their labels: Two nodes with labels (w_1, c_1, d_1) and (w_2, c_2, d_2) receive the same initial color iff $c_1 = c_2, w_1 = w_2$ and $d_1 = d_2$. All nodes in U receive the same color, which is different from the colors of the nodes in V. We then run the color-refinement algorithm on G, which outputs a stable bi-coloring $C_1^U, \ldots, C_p^U$ for the nodes in U and $C_1^V, \ldots, C_q^V$ for the nodes in V. To create the lifted factor graph $G' = (U', V', E')$, we first create a lifted variable node u'_k for every color class C_k^U and a lifted factor node v'_l for every color class C_l^V. Each lifted variable node u'_k and lifted factor node v'_l corresponds to a set of edges in G, namely $E_{kl} = \{e_{ij} \in E : v_i \in C_l^V, u_j \in C_k^U\}$. If E_{kl} is non-empty, we connect the nodes u'_k and v'_l in G' by an edge with the weight $(\sum_{(i,j):e_{ij} \in E_{kl}} z_{ij})/|C_l^V|$. Let $\mathcal{I} = \{i : v_i \in C_l^V\}$ and (w, c, d) be the label of some $v \in C_l^V$. We label the node $v'_l \in V'$ by the tuple $(\sum_{i \in \mathcal{I}} w_i, c, d)$.

Example 4. *The output coloring of the color refinement algorithm is shown in Fig. 3.1b. According to this coloring, the variables are partitioned into sets* $\Big\{\{y_1\}, \{y_2, y_4\}, \{y_3\}\Big\}$ *and the factors are partitioned into sets* $\Big\{\{\phi_2\}, \{\phi_1, \phi_3\}, \{\phi_4\}\Big\}$*. From the color classes of Example 3, we obtain the lifted factor graph of Fig. 3.1c which represents the function:* $\mathrm{E}(\boldsymbol{y^*}) = 5\max(-y_1^* + 2y_2^* - 1, 0)^2 + 10\max(y_1^* - y_2^*, 0)^2 + 5\max(-y_3^* + 1, 0)^2$.

As noted before, the expression in the above example is essentially a weighted sum of hinge functions which has the exact same form as (2.1) defined for HL-MRFs. This implies that the function represented by the lifted factor graph can also be solved using ADMM. To map the solution of lifted problem back to the original space, we only need to assign the value of the representative variables of each lifting color class to all the variables in that color class.

3.4.2 Correctness of the method

We now show that optimizing over the lifted function produces the same objective value as optimizing the original function, and the optimal values of the variables in the original problem can be derived from their lifted counterparts. Our proof is based on an existing procedure for lifting the *Quadratic Programming* (QP) problems [108].

We show that the MAP inference problem in HL-MRFs can be cast as a QP problem and that lifting this problem produces another QP which is equivalent to the function produced by our lifting method.

To write the objective function of the MAP inference in Equation 2.13 as a QP, we replace the $\max$ functions by constraints over auxiliary variables ψ_i:

$$\min \sum_i w_i \psi_i^{d_i} \quad \textit{s.t.,} \;\; \psi_i \geq \sum_j z_{ij} y_j - c_i \, \forall i, \quad \boldsymbol{y}, \boldsymbol{\psi} \geq \mathbf{0} \tag{3.2}$$

QP problems are lifted by performing the color refinement algorithm on a graph called the *coefficient graph*. We will now explain how to construct the coefficient graph for (3.2) (for further details we refer to [108]). The coefficient graph of (3.2) is the 4-tuple $(U, V, \tilde{\Psi}, E)$ where the nodes $u_j \in U$, $v_i \in U$, and $\tilde{\psi_i} \in \tilde{\Psi}$ correspond to variable y_j, constraint i, and variable ψ_i, respectively. For each nonzero coefficient z_{ij} there is an edge with weight z_{ij} between the nodes u_j and v_i. For each constraint i, the nodes $v_i \in V$ and $\tilde{\psi_i} \in \tilde{\Psi}$ are connected by an edge with weight $-c_i$, and if $d_i = 2$ then there is a self-loop edge on $\tilde{\psi_i}$ with the weight w_i.

Initially all nodes in $U \cup \tilde{\Psi}$ have the same color, which is not shared by any node in $\tilde{\Psi}$. Two nodes $v_{i_1}, v_{i_2} \in V$ receive the same initial color iff $c_{i_1} = c_{i_2}$.

After performing the color refinement algorithm on this coefficient graph, the coefficient graph of the lifted QP problem is constructed by grouping the variables and constraints of each color class together. The edge weights are aggregated in the same way as previously described in our method. The optimal value of a variable in the original QP is equal to the optimal value of its lifted counterpart.

Example 5. *The coefficient graph of the QP corresponding to Example 2 and the coloring assigned to it by the color refinement algorithm is presented in Fig. 3.2.*

Assume that a function $\mathbf{E}$ is lifted to another function $\mathbf{E}'$ using our proposed method.

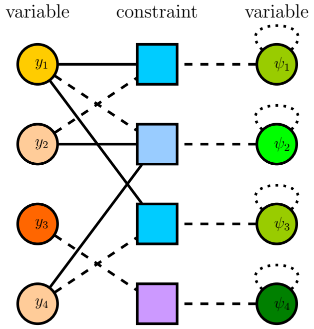


Figure 3.2: The colored coefficient graph of the HL-MRF model presented in Example 1. Edge weights are represented by line style (solid: 1, dashed: -1, dotted: 5).

We demonstrate the correctness of our method by showing that the QP of $\mathbf{E}$ can be lifted to the QP of $\mathbf{E}'$.

Theorem 1. *Let $G = (U^G, V^G, E^G)$ be the factor graph of an energy function of an HL-MRF, and $Q = (U^Q, V^Q, \tilde{\Psi}^Q, E^Q)$ be the coefficient graph of its QP. Then in the stable bi-colorings of G and Q, the color classes of U and V are the same.*

Proof. Assume that in the stable bi-coloring $\mathcal{C}^G$ of factor graph G, the nodes are partitioned into disjoint colors $C_1^G, \ldots, C_q^G \subseteq V^G$ and $C_{q+1}^G, \ldots, C_{q+p}^G \subseteq U^G$. Let us denote by u_j^G and v_i^G the nodes in G corresponding to variable y_j and factor ϕ_i in the HL-MRF energy function. Also let u_j^Q, v_i^Q and $\tilde{\psi}_i^Q$ denote the nodes corresponding to variable y_j, constraint i, and auxiliary variable ψ_i in the corresponding QP problem. We will now construct a stable bi-coloring $\mathcal{C}^Q$ of the coefficient graph Q with the following properties: 1) Variable nodes have the same color classes in $\mathcal{C}^Q$ and $\mathcal{C}^G$ and constraint nodes in $\mathcal{C}^Q$ have the same color classes as factor nodes in $\mathcal{C}^G$, 2) $\mathcal{C}^Q$ is consistent with the initial coloring of Q, and 3) $\mathcal{C}^Q$ is the coarsest stable bi-coloring of the graph Q. We first construct the coloring $\mathcal{C}^Q$ and then show how the above conditions hold for it. Let $\mathcal{C}(u)$ denote the color class of u in the coloring $\mathcal{C}$. In $\mathcal{C}^Q$ we assign the colors to the nodes in U^Q, V^Q, and $\tilde{\Psi}^Q$ based on the color classes of U^G and V^G in $\mathcal{C}^G$ as follows:

$$u^Q_{j_1} \in \mathcal{C}^Q(u^Q_{j_2}) \Leftrightarrow u^G_{j_1} \in \mathcal{C}^G(u^G_{j_2}) \tag{3.3}$$

$$v^Q_{i_1} \in \mathcal{C}^Q(v^Q_{i_2}) \Leftrightarrow v^G_{i_1} \in \mathcal{C}^G(v^G_{i_2}) \tag{3.4}$$

$$\tilde{\psi}^Q_{i_1} \in \mathcal{C}^Q(\tilde{\psi}^Q_{i_2}) \Leftrightarrow v^G_{i_1} \in \mathcal{C}^G(v^G_{i_2}) \tag{3.5}$$

The first property holds by definition. By definition, the nodes $u^Q_{j_1}, u^Q_{j_2} \in U^Q$ have the same initial color iff the initial colors of the nodes $u^G_{j_1}, u^G_{j_2} \in U^G$ are the same. Similarly, $v^Q_{i_1}, v^Q_{i_2} \in V^Q$ receive the same initial color iff the nodes $v^G_{i_1}, v^G_{i_2} \in V^G$ have the same initial color. Additionally, all nodes in $\tilde{\Psi}^Q$ receive the same initial color. Hence $\mathcal{C}^Q$ is consistent with the initial coloring of Q.

To show that the coloring is stable, we need to show that the sum of edge weights connecting to the nodes in each color class is the same among all the nodes having the same color. So for each pair of variable nodes $u^Q_{j_1}, u^Q_{j_2} \in U^Q$ and color class C^Q_l it should hold that $u^Q_{j_1} \in \mathcal{C}^Q(u^Q_{j_2}) \Leftrightarrow \sum_{i:v^Q_i \in C^Q_l} z_{ij_1} = \sum_{i:v^Q_i \in C^Q_l} z_{ij_2}$. Since $\mathcal{C}^G$ is a stable coloring of G we have $u^G_{j_1} \in \mathcal{C}^G(u^G_{j_2}) \Leftrightarrow \sum_{i:v^G_i \in C^G_l} z_{ij_1} = \sum_{i:v^G_i \in C^G_l} z_{ij_2}$ which together with equation 3.3 proves this property. Similarly, for each pair of constraints i_1, i_2 and color class C^Q_k it should hold that $v^Q_{i_1} \in \mathcal{C}^Q(v^Q_{i_2}) \Leftrightarrow \sum_{j:u^Q_j \in C^Q_k} z_{i_1 j} = \sum_{j:u^Q_j \in C^Q_k} z_{i_2 j}$ which can be concluded from equation 3.4 and the fact that $v^G_{i_1} \in \mathcal{C}^G(v^G_{i_2}) \Leftrightarrow \sum_{j:u^G_j \in C^G_k} z_{i_1 j} = \sum_{j:u^G_j \in C^G_k} z_{i_2 j}$. Note that the weights of edges connecting to nodes of ψ_i are not included in these equations since ψ_i variables appear with the same coefficient in all constraints. For a pair of nodes $\tilde{\psi}^Q_{i_1}, \tilde{\psi}^Q_{i_2} \in \tilde{\Psi}^Q$ where $d_{i_1} = d_{i_2} = 1$, we should have $\tilde{\psi}^Q_{i_1} \in \mathcal{C}^Q(\tilde{\psi}^Q_{i_2}) \Leftrightarrow v^Q_{i_1} \in \mathcal{C}^Q(v^Q_{i_2})$ which trivially holds according to equation 3.5. Finally, when $d_{i_1} = d_{i_2} = 2$, it should hold that $\tilde{\psi}^Q_{i_1} \in \mathcal{C}^Q(\tilde{\psi}^Q_{i_2}) \Leftrightarrow v^Q_{i_1} \in \mathcal{C}^Q(v^Q_{i_2}) \wedge w_{i_1} = w_{i_2}$ which holds according to equation 3.5 and the fact that if $w_{i_1} \neq w_{i_2}$ then the nodes $v^Q_{i_1}, v^Q_{i_2} \in V^Q$ are initialized with different colors.

Now what remains is to show that $\mathcal{C}^Q$ is the coarsest stable coloring of graph Q, i.e., there is not another stable bi-coloring respecting the previous conditions that assigns fewer number of colors than $\mathcal{C}^Q$ to the nodes of Q. Assume that there is a stable bi-coloring $\mathcal{C}'^Q$ of Q with fewer colors than $\mathcal{C}^Q$. Then we can construct a stable bi-coloring $\mathcal{C}'^G$ for the factor graph G that respects its initial coloring, by partitioning the U^G and V^G according to the color classes of U^Q and V^Q in $\mathcal{C}'^Q$. Since partitions of U^Q and $\tilde{\Psi}^Q$ are in one-to-one correspondence, the reduction in the number of color classes in $\mathcal{C}'^Q$ can not be limited to color classes of the nodes in $\tilde{\Psi}^Q$. This means that $\mathcal{C}'^G$ has fewer color classes than $\mathcal{C}^G$, which is a contradiction. □

3.5 Empirical Evaluation

In this section, we evaluate our proposed lifted inference algorithm, LHL-MRF, on various real and synthetic datasets. We investigate three research questions in our experiments: **Q1:** How does lifting affect performance on real world datasets? **Q2:** How does the graph structure influence the impact of lifting? **Q3:** How much symmetry is required for lifting HL-MRFs to be effective? All experiments were run on a machine with 16GB RAM and an i5 processor. The implementations are all single-threaded. We implemented our models using the PSL open-source Java library.[2] We ground the rules using the PSL library and then run inference using our own implementation of ADMM in C++.[3] Note that PSL removes a large number of trivial symmetries during the grounding process by removing trivially satisfied rules (for further information see [9]). Removing these simple symmetries ensures the extra symmetries that are obtained during our approach are non-trivial. We use Saucy[4] from the *RELOOP* library to perform color refinement [105].

[2] https://github.com/linqs/psl

[3] https://github.com/linqs/srinivasan-aaai19

[4] http://vlsicad.eecs.umich.edu/BK/SAUCY

Experiments on Real-world Data

We selected three real world datasets from different domains for which the corresponding PSL models have been used with promising results.

-**Citeseer**: This dataset includes 3312 papers in six categories, and 4591 citation links. The goal is to classify documents in a citation network. The original data comes from Citeseer . The details about the model and data can be found in [11].

-**Cora**: This dataset includes includes 2708 papers in seven categories, and 5429 citation links. The goal is to classify documents in a citation network. The original data comes from Cora . The details about the model and data can be found in [11].

-**Wikidata**: The dataset contains 419 families and 1,844 family trees. The goal is to perform entity-resolution on a family graph obtained form wikidata by crawling the site for familial relations. The details about the model and data can be found in [85].

To address Q1, we measure the effects of lifting on the three datasets. Figure 3.3 presents the number of variables and potentials of these datasets before and after lifting. We observe that there is a varying amount of symmetry in these datasets, the reduction in number of variables and potentials is about 20% in the Wikidata, 46% in the Cora, and 66% in the Citeseet dataset.

Datasets	HL-MRF (in sec)	LHL-MRF (solving) (in sec)	LHL-MRF (lifting) (in sec)	LHL-MRF (total) (in sec)
Citeseer	57.4	19.8	0.39	20.19
Cora	47.7	17.5	0.53	18.03
Wikidata	636.0	463.7	112.7	576.4

Table 3.1: Time taken to perform inference on different datasets.

Table 3.1 shows the time to solve the original problem, i.e., HL-MRF, the time to solve the lifted problem, i.e., LHL-MRF (solving), the time to lift HL-MRF with the

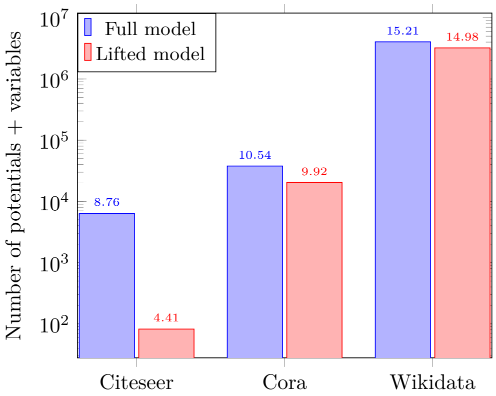


Figure 3.3: The number of variables and rules reduce by different amounts after lifting in real-world datasets.

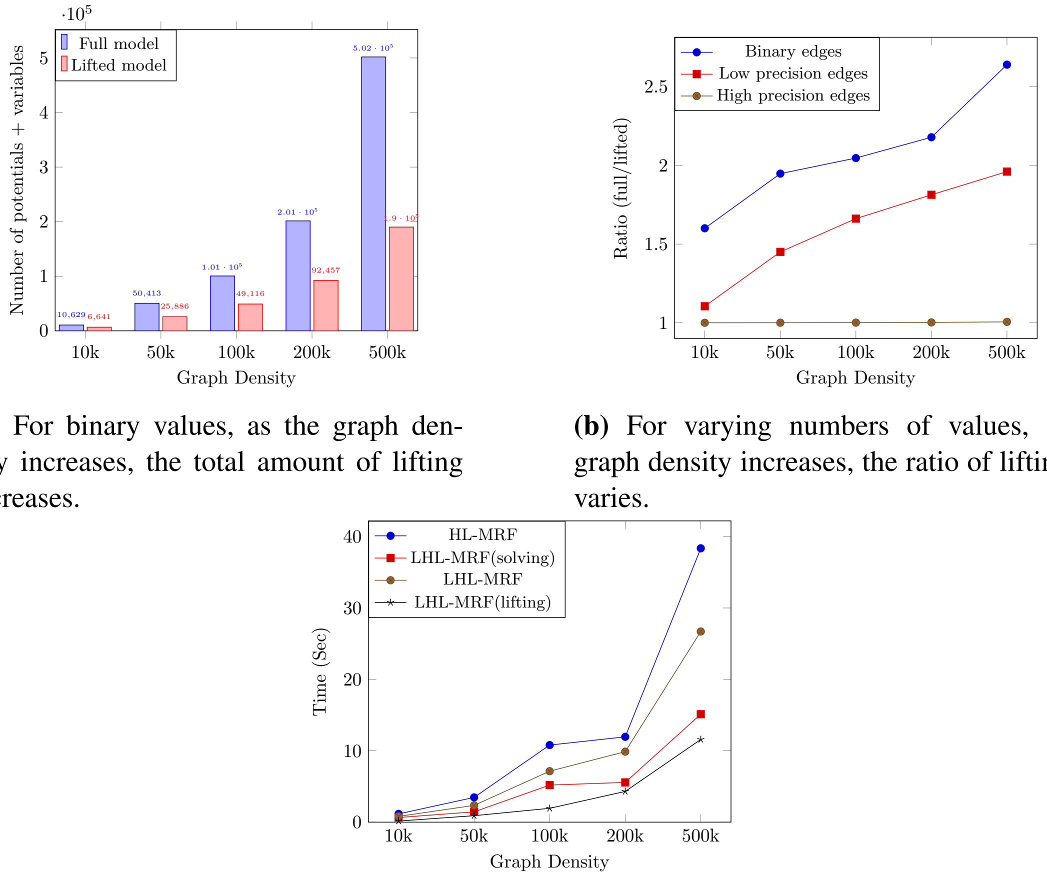
color refinement algorithm, i.e., LHL-MRF (lifting), and the end to end inference time for the lifted approach, i.e., LHL-MRF (total) or LHL-MRF in short.

As expected, due to the large amount of reduction in the number of variables and potentials, there is a significant difference between the time taken for HL-MRF and LHL-MRF (solving) on all subsets. Even with a small 20% reduction in number of variables and potentials in Wikidata, we see that LHL-MRF (Solving) is 27% faster than HL-MRF and due to much higher reduction in other datasets, we see three-fold speed-ups in both the Cora and the Citeseer datasets.

However, lifting time (LHL-MRF (lifting)) must also be considered. After accounting for this, we still see that LHL-MRF is about a 10% faster in the Wikidata dataset and almost three times faster for the Cora and the Citeseer datasets when compared to HL-MRF.

Experiments on Synthetic Data

To address Q2 and Q3 and better understand how symmetry is affected by graph density, we generate five different synthetic graphs. We also generate three sets of possible continuous values that the edges of the graph can take to generate different struc-



(a) For binary values, as the graph density increases, the total amount of lifting increases.

(b) For varying numbers of values, as graph density increases, the ratio of lifting varies.

(c) Comparison of inferences times for HL-MRF and LHL-MRF as graph density varies.

Figure 3.4: Comparison of inference times and sizes of the problem for HL-MRF and LHL-MRF as graph density varies

ture of neighbors in the graph. We generate the graphs for the task of node labeling with varying levels of density. We used a PSL model for the commonly used smoker example as introduced in [130], which describes smoking behavior among friends with the following rules:

$1.0 : Friend(A, B) \wedge Smokes(A) \rightarrow Smokes(B)$

$1.0 : Friend(A, B) \wedge \neg Smokes(A) \rightarrow \neg Smokes(B)$ This model states that if two people are friends, then either both of them smoke or neither of them do.

We fix the number of users to 1000, and randomly create friendship links between these users by varying the number of edges from $10k$ to $500k$. In practice friendship is not necessarily a black-and-white matter, i.e., people can be friends to varying degrees. Hence, we consider three cases for the values of the friendship links: 1) binary values, 2) values between zero to one with one decimal point, 3) values between zero to one with four decimal points. This means that friendship links can take only two values in the first case ($\{0, 1\}$), 10 values in the second ($\{0.0, 0.1, \ldots, 1.0\}$) and 10,000 in the third case ($\{0.0000, 0.0001, \ldots, 1.000\}$). We randomly assign a label to users and keep 50% of the labels as evidence and another 50% as unknowns to be inferred. Figure 3.4a shows the total number of variables and potentials before and after lifting for the binary case. Figure 3.4b shows the ratio between the number of variables and potentials before and after lifting, for varying value ranges. We see that for the binary case, the amount of lifting is maximized and the ratio increases as the graph density increases. However, as the value range increases, the amount of lifting drops significantly, and eventually there are no symmetries to be exploited. Finally, Figure 3.4c presents the processing time to solve the binary case. The results indicate that using LHL-MRF gives a significant performance improvement over HL-MRF as the graph becomes denser. These results imply that there are complex trade-offs between the structure of the graph and the range of the values in the data. We utilize exact lifting and therefore, we observe that LHL-MRF performs well for finding symmetries in datasets with denser structures and smaller range of values.

Finally, to further understand how the amount of symmetry affects the overall inference time in a slightly more complex and realistic setting (yet still a synthetic dataset), we study the social affiliation dataset and the PSL model used by [11] for scalability analysis. We use a dataset that contains 22k nodes and 130k edges. [5]

We begin by lifting this dataset to remove all symmetry (the original dataset has less than 1% symmetry). To induce symmetry, we systematically inject the same structure

[5] https://github.com/stephenbach/admm-speed-test

to the data. This is done by duplicating every grounded rule and shuffling the data. We duplicate the grounded rules up to 10 times creating 10 subsets (named 1x, 2x, 3x..., 10x), where the 10x subset has 10 times as many potentials created by duplicating the original data i.e., the 1x subset.

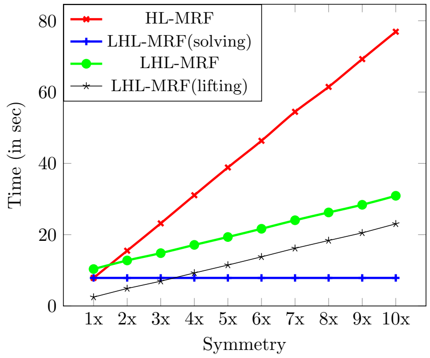


Figure 3.5: As symmetry increases, the time gap between solving HL-MRF and LHL-MRF increases.

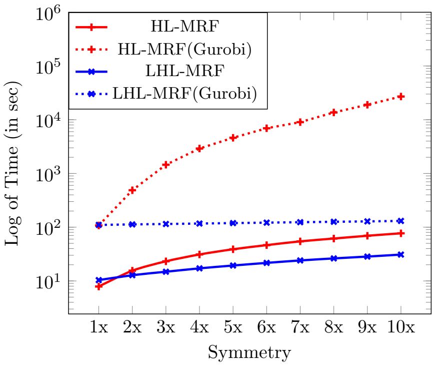


Figure 3.6: As symmetry increases, the time gap between solving HL-MRF and HL-MRF(Gurobi) increases exponentially. The difference between LHL-MRF and LHL-MRF(Gurobi) remains almost the same as the 1x dataset.

Figure 3.5 shows the results of HL-MRF and LHL-MRF (split into LHL-MRF (solving), LHL-MRF (lifting), and LHL-MRF on all 10 subsets. For the smallest dataset (which contains no symmetry), the time required for LHL-MRF is higher than HL-MRF due to the time taken to perform lifting. However, increasing the size of the dataset from two to ten, we observe that the amount of time taken by LHL-MRF to solve the problem is getting much lower than HL-MRF. It is noticeable that as the symmetry increases, the gap between solving HL-MRF problem and LHL-MRF problems widens. Note that the inference time in LHL-MRF for all 10 subsets is the same (equal to 1x dataset), which is the flat line in Figure 3.5 for LHL-MRF (solving).

For the sake of completeness and to compare against other lifted inference methods, in Figure 3.6, we compare the performance of HL-MRF and LHL-MRF using ADMM with versions which use Gurobi– an off-the-shelf commercial QP solver–. We denote these methods which use Gurobi HL-MRF (Gurobi) and LHL-MRF (Gurobi). For all 10 subsets, we observe that using HL-MRF and LHL-MRF consistently and significantly outperforms HL-MRF (Gurobi) and LHL-MRF (Gurobi) respectively. We also see that this difference increases as the size of the data increases. Note that the time taken to solve using LHL-MRF (Gurobi) is similar to other lifting methods such as belief propagation. For most of the lifting methods, the time complexity grows cubicly with the number of variables in the data. However, our approach is unique and desirable as it maintains the original form of the function allowing us to use ADMM which is known to be much more scalable than other approaches [48].

To our best knowledge, the size of datasets used in other lifted inference papers are in order of 1000s of variables and potentials, whereas using ADMM in our approach allow us to easily scale to problems with millions of variables and potentials.

3.6 Conclusion and Future work

In this paper, we introduced LHL-MRF, a novel approach to lifted inference in HL-MRFs. LHL-MRF marries the powerful ideas of lifted inference with the color refinement algorithm of [60] with the convex inference approach proposed by [11], to solve large-scale graphical models described by HL-MRFs. By combining these two ideas, our method is able to reduce the number of variables and potentials in a model and perform inference efficiently on a significantly smaller optimization problem. Through empirical evaluation, we show that the inference task for HL-MRF models on relatively small real world problems can be made to run three times faster. Further, in our experiments, we investigated how varying symmetry affects the performance of LHL-MRF and we explored the impact of both structure and domain values on the efficiency of LHL-MRF.

We have shown that there are significant opportunities for lifted inference, even in the case where we have continuous-valued variables defined by HL-MRFs. Through empirical evaluation on real datasets, we show that the inference task for HL-MRF models can run up to three times faster by using LHL-MRFs. However, it is important to note that LHL-MRFs cannot guarantee speed-ups for all types of problems. We investigate the effects of graph density and range of real values on lifting in HL-MRFs. However, studying the characteristics of the optimization problem after lifting is left for future work. We also notice that on small sized problem, in which inference takes less than one second to finish in HL-MRFs, the overhead of lifting is noticeable, and therefore even with a huge amount of reduction in the number of variables and potentials, we cannot necessarily reduce the solving time of LHL-MRFs.

This work suggests other interesting directions for future work. First, in this work we only exploit exact symmetries, which may be hard to find in some applications. Previous work indicates approximate lifted inference can improve the performance without compromising on other metrics like precision [135]. In our setting, approximate lifting could also lead to a greater reduction in number of variables and speed up the task of inference. Second, two of the most challenging tasks in MRFs are learning the weights and the structure of the logical rules from the data. Structure learning and weight learning are often performed using a scoring function that iteratively uses a MAP state. An interesting path to explore is to employ lifted inference to make such systems more efficient.

Chapter 4

Tandem Inference: An Out-of-Core Streaming Algorithm For Very Large-Scale Relational Inference

In this chapter[1] I discuss in detail an approach to scale inference in relational models (in specific PSL) to prohibitively large models that do not fit in the main memory of the machine. Further, I develop a novel streaming approach for both grounding and inference in PSL that make use of disk space and enables us to perform inference in tandem with model grounding. Through evaluation on realworld and synthetic datasets I show the effectiveness of our approach in performing extremely large scale inference in PSL.

4.1 Introduction

SRL methods have seen a great deal of success, they face a major challenge when scaling to large datasets. The benefit of easily generating large graphical models from a

[1] Appeared in *34th AAAI Conference on Artificial Intelligence (2020)*

few template rules can turn into a curse as the graphical models can quickly grow to intractable sizes that do not fit in memory. Consider a simple transitive rule: $Link(A, B) \wedge Link(B, C) \rightarrow Link(A, C)$, commonly used in conjunction with link prediction tasks. When this rule is instantiated (*grounded*) with a simple dataset of 1000 entries, a graphical model with a billion potentials is generated.

To address this issue, in this chapter, we propose an alternate approach to scaling which performs inference in tandem with grounding. This enables us to scale SRL systems to large, previously intractable, models. Our approach, which we refer to as *tandem inference* (TI), uses a novel streaming grounding architecture and an out-of-core inference algorithm that utilizes a disk cache in order to consume a fixed amount of memory. This allows TI to scale unbounded by a machine's main memory. Furthermore, even with increased I/O overhead, our approach runs the entire process of grounding and inference in a fraction of the runtime required by traditional approaches. Since TI is orthogonal to lifting and some of the other strategies, it can be combined with them for further improvements.

The TI concept is general and can potentially be applied to several different SRL frameworks. In this chapter, we show how it can be implemented in probabilistic soft logic (PSL) [11]. PSL is a SRL framework that generates a special kind of undirected graphical model called a hinge-loss Markov random field (HL-MRF). A key distinguishing factor of a HL-MRF is that it makes a continuous relaxation on random variables (RVs) which transforms the inference problem into a convex optimization problem. This allows PSL to use optimizers such as alternating direction method of multipliers (ADMM) [21] to perform efficient inference.

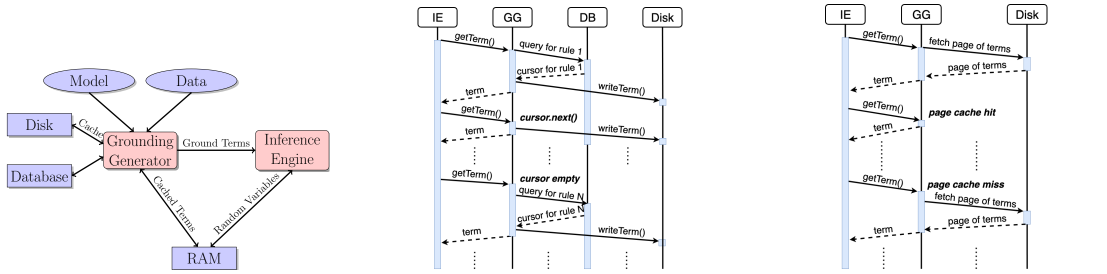
Our key contributions are as follows: 1) we propose a general framework, TI, which uses streaming grounding and out-of-core streaming inference to perform memory efficient, large-scale inference in SRL frameworks; 2) we derive a stochastic gradient descent-based inference method (SGD) and show that the SGD-based method can outperform the traditionally used ADMM-based method; 3) we develop an efficient streaming grounding architecture and SGD-based out-of-core inference system that runs faster than previous state-of-the-art systems; 4) through experiments on two large models, FRIENDSHIP-500M and FRIENDSHIP-1B, which require over 400GB and 800GB of memory respectively, we show that, using just 10GB of memory, we can perform inference on FRIENDSHIP-500M in under four hours and FRIENDSHIP-1B in under nine hours; and 5) we perform an empirical evaluation on eight realworld datasets to validate the speed and accuracy of TI. In addition to enabling inference on models too large to fit into memory, on our largest realworld dataset which does fit in memory, TI is 8 times faster than traditional approaches.

4.2 Related Work

Several approaches have been proposed to scale relational models to large datasets. *Lifted inference* [38, 140, 41, 70, 134, 78, 148] is a commonly employed and effective approach that exploits symmetry in the data to generate a more compact model on which to perform inference. While effective in many settings, a key drawback of these approaches is that evidence or noisy data can break symmetries, making lifting less effective. To address this issue, *approximate lifting* approaches have also been proposed [135, 40, 156, 33] which exploit approximate symmetries, allowing for greater and more robust compression. However, if the ground model lacks significant symmetry, approximate lifting may improve tractability only at the cost of correctness.

Several approaches orthogonal to lifted inference have also been proposed that attempt to perform efficient grounding by utilizing hybrid database approaches [116], exploiting the structure of rules [9], perform efficient approximate counting for faster inference [157, 133, 34], exploiting sparse structure in data to constrain the size of models (blocking) [117, 10] or distributing models across multiple machines [97]. However these methods, while quite useful, only provide partial solutions to grounding and efficient inference at scale. The hybrid database approach increases runtime substantially, exploiting rule structure requires large amounts of memory to store the ground model, approximating counting methods are applicable to discrete graphical models only, blocking needs high understanding of data and can constrain the types of models on can generate, and distributing across several machines does not reduce the overall memory required to run a large program.

4.3 Tandem Inference



(a) Block diagram showing the TI system architecture.

(b) Network sequence diagram for iteration 1 of TI.

(c) Network sequence diagram for iterations 2 through T of TI.

Figure 4.1: The architecture of TI.

In order to define our proposed tandem inference (TI) algorithm we introduce two components: the *grounding generator* (GG) and the *inference engine* (IE). The GG supports *streaming grounding*, which is the process of generating ground rules in small batches without materializing all grounding results into memory. The IE supports *streaming inference*, which is the process of performing inference using a single potential at a time. Fig. 4.1a shows the system architecture of TI. The GG takes as input the data $\mathcal{D}$ and the model $\mathcal{M}$, which GG uses to generate the ground model. With respect to storage, the GG can leverage the hard disk, the database, and RAM, while the IE can only utilize RAM.

The process flow of TI is shown in the network sequence diagrams given in Fig. 4.1b (for the first round of inference) and 4.1c (for subsequent rounds of inference). The IE begins by requesting a potential function (also called a *ground term*) from the GG. During the first round of inference, the GG utilizes the database to generate ground rules. Once a ground rule is created, it is converted into a ground term, written to a disk cache, and then passed onto the IE. On subsequent rounds of inference, the GG uses the disk cache alone to provide ground terms to the IE. As each term is returned from the GG, the IE will optimize the term and update any relevant RVs held in RAM. After all terms have been seen, IE will then begin a new round of inference until convergence criteria are met. Since the GG uses a fixed-size in-memory cache and the IE discards terms after use, there is a maximum amount of memory in use by TI at any given time.

4.3.1 Streaming Grounding

Streaming grounding is the TI component that is responsible for providing ground terms one at a time to the IE. To support streaming grounding, the underlying SRL framework must be able to construct a single ground rule without instantiating large portions of the model, a process we will refer to as *partial grounding*. Constructing the full ground network is the SRL phase that is most prone to running out of memory, so it is imperative that this process can be broken up into small chunks. PSL is one of several SRL frameworks that supports *bottom-up grounding* [9], which frames the problem of grounding a rule as constructing and executing a SQL query. Relational database management systems (RDBMSs) have a built-in way to fetch only a portion of the query results through cursors [50]. A cursor works as an iterator for a query's results. If the RDBMS and content of the SQL query allows for it, cursors can return results as they are generated by the query instead of waiting for the full query to complete. Cases that force a database to materialize all results before returning any records include sorting or deduplicating the results, both of which are avoided in PSL grounding queries.

During the initial iteration of streaming grounding, the database must be queried to fetch the ground terms. The process described here is also shown as a network sequence diagram in Fig. 4.1b. The rules in the model, $\mathcal{M}$, will be iterated over until all have been grounded. When asked for a term, the GG will first check if it has an open database cursor. If there is no cursor or the current cursor has been exhausted, then the database will be queried for the next rule and a new cursor will be constructed. If all rules have been grounded, then the GG will inform the IE that there are no more ground terms for this iteration. With an open cursor, the next result tuple will be fetched. The tuple will then be instantiated into a ground rule and checked for triviality. A ground rule is trivial if it will not affect the result of inference; for example, a ground rule with a logical expression that is already satisfied by observed variables is considered trivial. If the newly instantiated ground rule is invalid, the process is repeated with the next result from the cursor until a valid ground rule is instantiated. After a ground rule is validated, it is converted into a ground term. This term is then put into a cache buffer and eventually passed on to the IE. Once the cache buffer is full, or there are no more ground rules, it is written to disk.

After the initial iteration of TI, ground terms are fetched from the disk cache in the order they were written during the initial iteration of streaming grounding. The process described here is shown as a network sequence diagram in Fig. 4.1c. The disk cache is written as several pages, one page to a file. The number of terms written to a page can be configured and the effect of this configuration is explored in Section 4.4.2. When asked for a term, the GG will first ensure that a cache page is loaded. If the current page has been exhausted and there are no more pages, then the IE will be informed that there are no more ground terms for this iteration. Each page is written in binary to minimize I/O. The first 16 bytes of a page contains the number of terms written to the page and the size in bytes of all the terms in the page. Because each term may contain a different number of RVs, the exact size of each term is not known until it is generated. Each term is then read into a preallocated term structure. A free list of available terms large enough to fill a page is maintained to minimize memory allocations. After all terms have been read into the memory cache, the next term from the cache will be returned to the IE until the page is exhausted.

The GG also provides the ability to return ground terms in a semi-random order. The effectiveness of randomizing term order is dependent upon the optimization algorithm employed by the IE. During the initial iteration of streaming grounding, the order of the terms is dependent upon the order that results are returned from the database. However, during subsequent iterations, there are more opportunities to induce randomness. First, the order that pages are accessed can be randomized. Second, the terms in each page can be shuffled in-place after they have been read from disk. Although not fully random since every term is guaranteed to be no more than a page length away from other terms in the same page, these steps provide a good amount of randomness without increasing the number of I/O operations or memory required.

4.3.2 Streaming Inference

The second core component of TI is streaming inference. The open-source PSL implementation uses ADMM to minimize the convex objective produced by a HL-MRF. While ADMM is an efficient algorithm, it places a high memory requirement on the system. However, our primary goal of using TI is to alleviate any memory constraints for performing inference. ADMM in PSL works by creating Lagrange variables (LVs) and a local copy of the RVs (LRVs) for every potential. Every potential is optimized w.r.t. the LRVs and an average over all LRVs is taken to obtain a global assignment for the RVs. All ground rules, LRVs, and LVs are kept in memory to make this process efficient. However, the need to keep all this information in memory makes ADMM less than ideal for streaming inference. To replace ADMM, we propose a gradient-based optimization to solve Equation 2.13.

Stochastic Gradient Descent for PSL

Consider the energy function from Equation 2.1 defined for HL-MRFs. To facilitate easy gradient computation, this can be re-written as:

$$
\begin{aligned}
E(\mathbf{y}|\mathbf{c}) &= \sum_{i=1}^{K} w_i \phi_i(\mathbf{y}, c_i) \\
\phi_i(\mathbf{y}, c_i) &= \begin{cases} (\mathbf{y}^T \mathbf{q}_i - c_i)^{d_i} & \text{if linear loss} \\ \max((\mathbf{y}^T \mathbf{q}_i - c_i), 0)^{d_i} & \text{otherwise} \end{cases} \\
c_i &= \mathbf{x}^T \dot{\mathbf{q}}_i
\end{aligned}
\tag{4.1}
$$

where $\mathbf{q}_i \in \{0, 1, -1\}^n$ and $\dot{\mathbf{q}}_i \in \{0, 1, -1\}^m$ are respective n-dimensional and m-dimensional ternary vectors which indicates if a variable participates positively, negatively, or not at all in potential function ϕ_i. The scalar c_i incorporates the information of all the observed variables participating in ϕ_i and we use $\mathbf{c}$ to represent the vector of scalars c_i. As a reminder, in full gradient descent (GD), we iteratively optimize by taking weighted steps in the direction of the energy function's (Equation 4.1) gradient until convergence or for T steps. The weight of the steps is determined by the learning rate, η. Additionally for PSL, Equation 2.1 has a restriction that $\mathbf{y} \in [0, 1]^n$, therefore after each step we need to project $\mathbf{y}$ back in to the box $[0, 1]$ through truncation. The gradient step update at every step t can be represented as:

$$\mathbf{y}_t = \mathbf{y}_{t-1} - \eta \nabla_{\mathbf{y}} E(\mathbf{y}|\mathbf{c}) \tag{4.2}$$

$$\mathbf{y}_t = \min(\max(\mathbf{y}_t, 0), 1) \tag{4.3}$$

$$\text{where } \nabla_{\mathbf{y}} E(\mathbf{y}|\mathbf{c}) = \sum_{i=1}^{K} w_i \nabla_{\mathbf{y}} \phi_i(\mathbf{y}, c_i)$$

The gradient computation for the energy function involves computing projected gradients for the potential functions. The projected gradients can be written as:

$$\nabla_{\mathbf{y}} \phi_i(\mathbf{y}, c_i) = \begin{cases} 0 & \text{if hinge \& } \mathbf{y}^T \mathbf{x}_i \leq c_i \\ w_i \mathbf{q}_i & \text{if } d_i = 1 \\ 2 w_i \mathbf{q}_i (\mathbf{y}^T \mathbf{q}_i - c_i) & \text{otherwise} \end{cases} \tag{4.4}$$

Using the above equations, we can compute the gradient update and run the process to convergence. Since the objective is convex, the right choice of η will guarantee convergence. However, an issue with GD is that at every step it needs to compute the full gradient, requiring all terms. This is expensive and does not support our streaming approach. To make it more compatible with our streaming approach, we use stochastic gradient descent (SGD). In SGD, the gradient is computed w.r.t. a single potential ϕ_i and an update to the variables can be made without examining all terms.

$$\mathbf{y}_t = \mathbf{y}_{t-1} - \eta w_i \nabla_{\mathbf{y}} \phi_i(\mathbf{y}, c_i) \tag{4.5}$$

The variables are then projected as in Equation 4.3. This update is aligned with our streaming approach, and allows the IE to request a single potential, perform an update on the participating RVs, and continue on to the next potential.

An important factor to make SGD work in practice is the correct choice of η, and

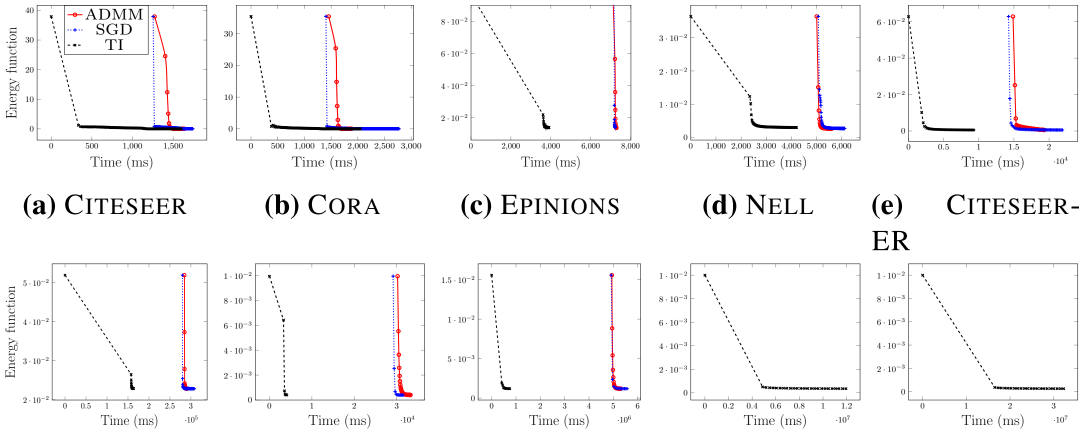
```
Algorithm 1: SGD for PSL
  Data: list of ground terms $\phi = \{\phi_1, \phi_2, ..., \phi_K\}$
  Result: RVs $\mathbf{y}$
  $\eta$ = learning rate;
  $\mathbf{y} \sim Unif(0, 1)^n$;
  $t = 1$;
  while not converged and $t <= T$ do
      for $i \in \{1 \dots K\}$ do
          Update $\mathbf{y}$ using $\phi_i$ with Equation 4.5;
          $\mathbf{y} = \min(\max(\mathbf{y}, 0), 1)$;
      end
      $t = t + 1$;
      Update $\eta$;
  end
```

many approaches have been proposed to compute adaptive learning rates [131]. One of the more popular and successful methods for adaptive learning rates is SGD-ADAM [79]. In this work we investigate three approaches: 1) using SGD-ADAM, 2) using a time-decaying learning rate (i.e., $\eta = \frac{\eta}{t}$), and 3) using a constant learning rate η. In our experiments, we observe that a time-decaying learning rate is more effective than more complicated mechanisms such as SGD-ADAM. Finally, the overall process of performing non-streaming inference using SGD in PSL is shown in Algorithm 1.

Now that we have a streaming grounding infrastructure and the SGD-based PSL IE, we can perform TI. The algorithm for TI is the same as Algorithm 1, except that the potentials are read from the GG instead of from memory.

4.4 Empirical Evaluation

In this section, we evaluate the performance of our proposed method on variety of realworld and two large synthetic datasets. We answer the following questions: Q1) Can we perform inference on large ground models that were previously intractable?



(a) CITESEER **(b)** CORA **(c)** EPINIONS **(d)** NELL **(e)** CITESEER-ER

(f) LASTFM **(g)** JESTER **(h)** JESTER-FULL **(i)** FRIENDSHIP-500M **(j)** FRIENDSHIP-1B

Figure 4.2: Comparison of the runtimes for TI, ADMM, and SGD on 10 datasets.

Q2) Is streaming faster than and as accurate as traditional inference? Q3) How much memory does TI use? Q4) Can a gradient-based optimizer converge faster than ADMM? Q5) What is the best strategy for selecting the learning rate? We answer Q1 and Q2 in Section 4.4.1, Q3 in Section 4.4.2, and Q4 and Q5 in Section 4.4.3. For all our experiments, we set the max number of iterations, T, to 500, a convergence tolerance of 10^{-6}, and a machine with 400GB of memory.

We perform our experiments on eight realworld datasets and two synthetic datasets from a data generator previously used to test scaling in PSL [9].[2] The details of the datasets are as follows:

CITESEER: a collective classification dataset with 2,708 scientific documents, seven document categories, and 5,429 directed citation links.

CORA: a collective classification dataset with 3,312 documents, six categories, and 4,591 directed citation links.

EPINIONS: a trust prediction dataset with 2,000 users and 8,675 directed links which represent positive and negative trust between users.

[2] Models, data, and code: https://github.com/linqs/aaai-ti

NELL: a knowledge graph construction dataset originally derived from the NELL project with 27,843 entity labels and 12,438 relations.

CITESEER-ER: an entity resolution dataset with a citation network of 1136 authors references and 864 paper references.

LASTFM: an artist recommendation dataset with 1,892 users, 17,632 artists, 92,834 user-artist ratings, and 12,717 friendship links.

JESTER: a joke recommendation dataset with 2,000 users, 100 jokes, and 200,000 user-joke ratings, sampled from the larger JESTER-FULL dataset.

JESTER-FULL: the full Jester dataset. Contains 73,421 users, 100 jokes, and 7.3M user-joke ratings. To the best of our knowledge, this is the first time the full Jester dataset has been used in with an SRL framework.

FRIENDSHIP-500M: a synthetic link prediction dataset with 2,000 users and 4M unobserved edges.

FRIENDSHIP-1B: similar to FRIENDSHIP-500M containing 2,750 users and 7.5M unobserved edges.

Table 4.1 provides details on number of rules in each model, the number of ground rules generated, and the amount of memory required to hold each model in memory.

Dataset	Rules	Ground Rules	Random Variables	Memory (GB)	Source
CITESEER	10	36K	10K	0.10	[11]
CORA	10	41K	10K	0.11	[11]
EPINIONS	20	14K	1K	0.12	[11]
NELL	26	91K	24K	0.13	[124]
CITESEER-ER	9	541K	485K	0.24	[19]
LASTFM	22	1.4M	18K	0.45	[84]
JESTER	7	1M	50K	0.49	[11]
JESTER-FULL	8	110M	3.6M	110	[57]
FRIENDSHIP-500M	4	500M	4M	400+	[9]
FRIENDSHIP-1B	4	1B	7.6M	800+	[9]

Table 4.1: Details of models used and their memory consumption for non-streaming inference. Memory usage for FRIENDSHIP-500M and FRIENDSHIP-1B are estimates. The memory consumed by TI depends on the page size chosen. In our comparison experiments, we use a page size of 10M which uses about 10GB of memory.

4.4.1 Scale, Speed, and Convergence

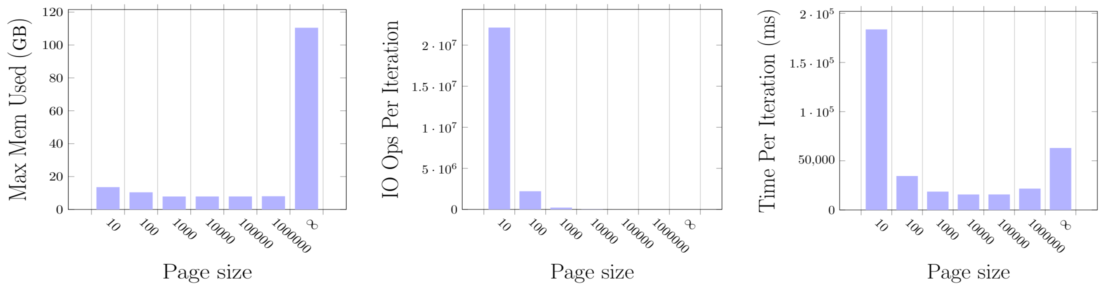
We begin by examining the inference time and convergence of TI, SGD, and ADMM on all ten datasets (Q1 & Q2). Weights for the rules in each model are learned and rescaled to be in the range $[0, 1]$. Both SGD and TI use a time-decayed learning rate and the initial learning rate η needs to be tuned. For the LASTFM, FRIENDSHIP-500M, and FRIENDSHIP-1B datasets we use $\eta = 0.1$, for CITESEER-ER we use $\eta = 10$, and for all the other datasets we use $\eta = 1.0$. The rational for choosing these learning rates is explained in Section 4.4.3. TI also has a page size which can be tuned based on the amount of memory available. Since our machine has 400GB RAM, we choose a page size of 10M for all datasets, which uses about 10GB of memory. We discuss further details about trade-offs in page size, memory, and computation in Section 4.4.2.

Scaling to Large Datasets:

Fig. 4.2i and 4.2j show the inference convergence w.r.t. time (in milliseconds) for FRIENDSHIP-500M and FRIENDSHIP-1B. TI was able to run the FRIENDSHIP-500M dataset in under four hours using only 10GB of memory. Both SGD and ADMM exhausted the 400GB of memory available on the machine and failed to run. Similarly, we observe that the FRIENDSHIP-1B dataset, which we estimate to require more than 800GB to hold in memory, was able to run on the same machine in under nine hours using only 10GB of memory. These results answer Q1 affirmatively, TI can successfully perform inference on large ground models that were previously intractable.

Speed and Convergence:

In order to address Q2, Fig. 4.2 shows the inference convergence for all datasets using all three approaches. The time shown includes both the grounding and inference phases (which happen together in TI). In all datasets except CITESEER and CORA,



(a) Maximum memory usage for TI over multiple page sizes.

(b) Number of I/O operations per optimization iteration of TI over multiple page sizes.

(c) Runtime per optimization iteration of TI over multiple page sizes.

Figure 4.3: Memory usage, I/O usage, and speed of TI on the JESTER-FULL dataset w.r.t. page size. Page sizes listed as ∞ are run with SGD, which does not use pages.

we observe that TI converges before the other methods even fully finish grounding! In CITESEER and CORA, possibly due to some rules with high weights, tuning the learning rate is difficult, and, after the first steep drop, SGD and TI take many more iterations to converge compared to ADMM. As the ground model size increases, we observe more significant timing differences. In EPINIONS, CITESEER-ER, and LASTFM, we see that TI finishes the entire process of grounding and inference in just half the time taken by ADMM and SGD. For JESTER, TI is over 5 times faster than both ADMM and SGD. For our largest realworld dataset, JESTER-FULL, TI is about 8 times faster than both ADMM and SGD. This shows that TI is faster than traditional approaches especially on larger datasets and converges to the same function value.

4.4.2 Memory Efficiency

To answer Q3, how much memory TI uses, and test the effect of page size, we run TI on the JESTER-FULL dataset with page sizes between 10 and 1M potentials. We run SGD to establish baseline behavior. Since SGD holds all components in memory, we consider it to have an infinite page size. We measure the maximum memory usage during the entire run, the mean number of I/O operations performed in a single iteration of optimization, and the mean time to complete a single iteration of optimization. Because PSL is written in Java, the memory usage we report is the size of the JVM's heap. I/O operations are measured by the number of calls made to Java's low-level I/O methods `FileInputStream.read` and `FileOutputStream.write`. All reported values are averaged over 10 runs.

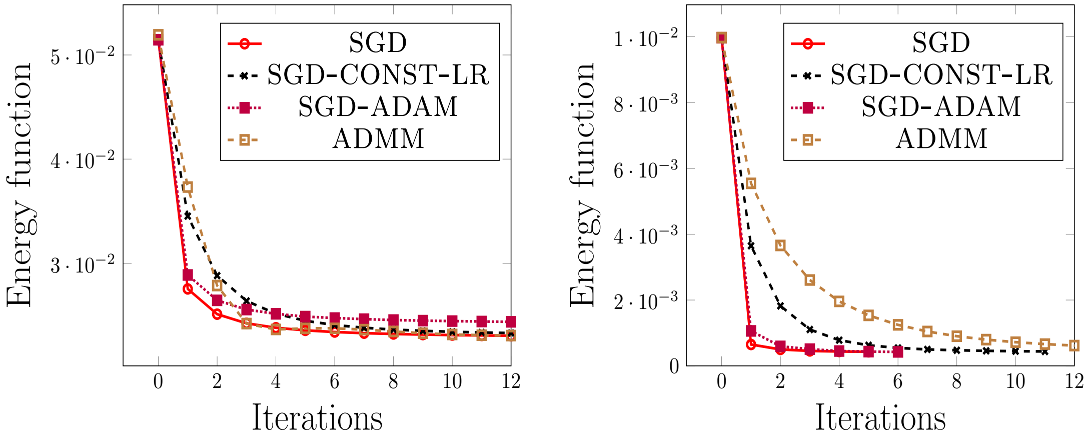
Fig. 4.3a shows the peak memory usage over several runs using different page sizes for TI. Naively, we expect the amount of memory used to decrease with the page size: the fewer ground terms held in memory, the less overall memory is used. However, instead we see that very small pages sizes (10 and 100) lead to more memory being used. To understand why, we must remember that Java is a garbage collected language and that memory marked for garbage collection will still count as being in use. The small page sizes cause many I/O operations to happen in quick succession. Every I/O operation requires Java to allocate memory buffers used in that operation. Therefore, the discarded buffers are eventually cleaned up but not before being counted in the memory total. A native language like `C` that can directly make system calls and that does not have to go through a virtual machine can avoid these extra allocations and inflated memory cost. Forcing the JVM to garbage collect more frequently[3] shows a more consistent memory usage over all page sizes. This is because all the page sizes still fit into memory and Java will continue to accumulate memory until a point where garbage collection is triggered. This point depends on the maximum size of the heap and is common among all the runs. From this experiment we can conclude that although using a smaller page size will use less persistent memory, the JVM's garbage collector will keep most reasonable page sizes at the same memory usage levels.

Fig. 4.3b shows the number of I/O operations per optimization iteration. SGD does not perform any I/O operations. As the page size increases, the number of I/O opera-

[3] A smaller value for the JVM parameter `XX:NewRatio` is used to force more frequent garbage collection.

tions decreases. The impact of these additional I/O operations for a small page sizes can be seen in Fig. 4.3c (per-iteration runtime). The different page sizes result in approximately the same number of bytes read from disk and since the number of potentials is the same, the time spent in optimization will also be the approximately the same. However, the overhead of the additional I/O operations causes substantially slower iterations for page sizes of 10 and 100. For larger page sizes (1000+), the difference in I/O overhead becomes negligible and all have similar per iteration runtime.

4.4.3 Optimizer Efficiency and Learning Rate



(a) LASTFM dataset.

(b) JESTER dataset.

Figure 4.4: The effect of different optimizers on convergence.

To answer Q4, can SGD converge as fast as ADMM, and Q5, how to choose learning rate, we run experiments on the LASTFM and JESTER datasets. The results here extend to other datasets. Here, we compare four different approaches: SGD with a decaying learning rate (SGD), SGD with a constant learning rate (SGD-CONST-LR), SGD with an adaptive learning rate (SGD-ADAM), and ADMM.

SGD vs. ADMM:

Fig. 4.4 shows the convergence of different approaches w.r.t. number of iterations for the LASTFM and JESTER datasets. Here, we observe that SGD and ADMM converge to the same function value in different number of iterations. Typically, SGD takes fewer iterations to converge than ADMM. However, this is heavily dependent on the learning rate chosen for SGD. If one cannot find the right learning rate, then it is possible for SGD to take significantly more iterations than ADMM.

Choice of Learning Rate:

From Fig. 4.4 we observe that SGD-CONST-LR is the slowest to converge, and there seems to be little difference between SGD which uses time-decayed learning rate and SGD-ADAM which uses an adaptive learning rate. SGD and SGD-CONST-LR have an initial learning rate to be chosen. We observed that this can be chosen in range $\eta \in [\frac{1}{10\max(\mathbf{w})}, \frac{10}{\max(\mathbf{w})}]$ for SGD, and for SGD-CONST-LR, $\eta \in [\frac{1}{100\max(\mathbf{w})}, \frac{100}{\max(\mathbf{w})}]$. Thus, we choose an η of 1.0 and 0.1 for JESTER and LASTFM respectively for SGD, and η of 0.01 for SGD-CONST-LR. SGD-ADAM has four hyperparameter α, $\beta 1$, $\beta 2$, and ϵ to tune. This makes it harder to get ideal performance with SGD-ADAM. Further, SGD-ADAM uses additional parameters equal to three times the number of RVs to perform adaptive tuning. In our experiments, we choose $\alpha = 0.01$, and use $\beta 1 = 0.9$, $\beta 2 = 0.999$, and $\epsilon = 10^{-8}$ as suggested by [79]. From our evaluation we conclude that the simpler strategy, SGD with decaying learning rate performs just as well as SGD-ADAM, the more complicated adaptive strategy.

4.5 Conclusion and Future Work

In this chapter we introduce tandem inference, TI, a new out-of-core method for performing inference on large ground models that don't fit into main memory. To make TI possible, we introduce a streaming method for both grounding and inference. Through experiments on ten datasets, we have shown that TI can not only reduce runtime by up to eight times, but it can do so using a fixed amount of memory. The fixed memory nature of TI enables the SRL community to scale to problems that were previously unreachable.

While this chapter introduces the fundamentals of TI, there remain several areas for research. Incorporating lifted inference is a promising extension to TI. Because TI is orthogonal to lifting, these two can be combined to speed up inference further. Next, despite impressive performance on large datasets, the overall process of TI is largely sequential; parallelizing TI can be another way to speeding up inference further. Another interesting avenue for research is to create a hybrid IE using ADMM and SGD. SGD often minimizes quickly during the first few iterations, however may take many more iterations to fully converge (especially if the learning rate is poorly selected). Conversely, ADMM converges more slowly than SGD, but more steadily. A hybrid IE could start with SGD and then switch to ADMM after the first few iterations. Finally, TI can be extended to any other SRL framework that can support streaming grounding and streaming inference.

Chapter 5

Real-Time Structured Prediction Using PSL

In this chapter[1] I introduce a new task of collective classification in a product retrieval application. Further, I propose the concept of a *micrograph*, which generates a small graphical model for real-time structured prediction. We also develop a novel new inference engine for PSL that can significantly speed up inference and perform structured prediction in real-time. Through evaluations on realworld datasets, I show the accuracy of our approach in performing collective classification and efficiency of the new inference engine at speeding up inference.

5.1 Introduction

When customers shop online, they issue queries that describe their intent along multiple facets of their desired product: brand, color, product-type, age group, size, gender, and activity. For example, the query "red adidas shorts for boy age 6" has age, gen-

[1] Appeared in *28th ACM International Conference on Information and Knowledge Management (2019)*

der, color, brand and a product type specified. These facets are critical for matching, ranking, and navigation. For instance, in the case of navigation, a customer might first look for a specific brand and then narrow down their choices based on color, obtaining relevant results that align closely with their intent. However, due to past behavioral associations (such as clicks, purchases, and cart-adds) or noisy lexical information (such as low-quality seller supplied keywords), or competition between brands, the search might result in mismatched products.

Consequently, identifying facet mismatches between a query and products in the catalog to avoid displaying irrelevant results is an important component of providing customers with a satisfying shopping experience. A typical model for recognizing facet mismatches outputs a score for one or more facets given a query-product pair. This score indicates whether the product is a good match for the query along that specific facet.

In modern datasets, there are vast amounts of additional structural information about queries, products, and their relationships. This additional information can manifest itself in several ways and can be used as side information during the retrieval, ranking and mismatch classification training process.

1. Products are typically co-purchased or co-viewed together. We can include this information as a product-product graph.

2. We can generate query and product latent representations (embeddings) and use cosine similarity as an affinity score between (query, product), (query, query), and (product, product) pairs.

3. Customer query reformulations within the same session can be used to compute query-query similarities.

Incorporating such structural side information typically yields a boost in model performance. Indeed, learning with side information has shown to be successful in several applications such as recommender systems [84, 128], knowledge graphs [100], entity resolution [118], computer vision [61], and has recently been applied even in deep learning tasks [165, 161].

However, using additional information and performing graphical model inference is slower than performing inference using a simple pointwise binary classifier for facet mismatch. This presents an additional challenge for applying these ideas to product search, which requires the use of models with extremely fast inference for real-time retrieval of search results. Existing scalable approaches for graphical model inference [45, 151] do not meet the latency constraints of search systems. A customer may abandon the search if it takes more than a few milliseconds between typing in a query and obtaining the results. This constraint makes it extremely difficult to use additional side information or use sophisticated models with higher computational complexity.

In this chapter, we develop a novel approach using the PSL framework for facet mismatch classification and apply it to the task of detecting product-type mismatches, a particularly egregious form of facet mismatch since they lead to a significantly degraded customer experience. As an example, a customer searching for an “iPhone” expects to see different variety of iPhones in the search results and not an iPhone case or a screen protector. We show that incorporating additional structural information present in the data can significantly improve the classification performance. Secondly, to tackle the problem of near real-time inference, we cast the problem of PSL optimization as minimizing an SVM-like objective function and use a *Trust Region Quasi-Newton* (TRON) [93] method to solve it. We show that the resulting method achieves orders of magnitude speedups over existing approaches for PSL optimization. Note that the approach we propose is quite general and can be applied to many different label propagation applications, but in this chapter we focus on detecting product-type mismatches in search.

5.1.1 Contributions and Organization

To summarize, the contributions of our chapter are as follows:

- We introduce a special query-product relationship graph that we refer to as a *micrograph* which we show can be used to improve facet mismatch classifiers. Micrographs ensure that our approach scales independently of the number of queries, allowing us to use it for industry-sized datasets.

- We show how micrographs can be utilized in the PSL framework to improve facet mismatch classification by performing collective inference. We refer to this approach as *structured mismatch classification* (SMC). We also show that naive inclusion of structure does not improve the model performance significantly. Further, we introduce a variant of SMC which we refer to as *strong* SMC ($\mathrm{S}^2\mathrm{MC}$) which selectively performs joint inference to improve overall mismatch identification.

- We perform extensive experiments across multiple datasets and show that the method we propose improves upon baseline methods in performance by up to a *12%* increase in precision and an *11%* increase in F1 scores.

- We reformulate the resulting optimization problem which enables us to perform near real-time inference using quasi-Newton methods. Through a series of experiments, we show that our approach is scalable and can be used to make real-time predictions. Our approach of using a quasi-Newton method yields up to *150x speedup* over the existing solver (ADMM).

The rest of the chapter is organized as follows. In Section 5.2, we briefly discuss some work that incorporate structure in information retrieval tasks. Next, in Section 5.3, we formally set up the problem and discuss traditional solutions to the problem. Next in Section 5.4, we introduce the concept of micrographs, elaborate on them and show how they can be used in our problem setting. In Section 5.5 we define our approach on using micrographs to perform collective inference. Next, in Section 5.6, we discuss the need for extremely fast inference and show how we can efficiently make predictions using trust region Newton methods, which yields orders-of-magnitude speedups over existing ADMM solvers. We perform extensive experiments on multiple datasets and their results in Section 5.7. Finally, we summarize and conclude the chapter in Section 5.8.

5.2 Related Work

Many information retrieval [101, 26, 83, 5] and ranking [164, 166] tasks have used the structural information to improve their models. These approaches create a graph (or similar relational structure) using an item's lexical information and improve the list of items retrieved for a specific query. However, in this work, we do not focus on using structural information to retrieve a better list of items, rather we show a way to improve the quality of the retrieved list by identifying mismatched items. Our approach focuses on using heterogeneous structural information to identify search mismatches which can be subsequently used to either reorder or improve the list by replacing the mismatched items.

5.3 Problem definition and traditional approach

Our task is to improve search results by identifying the products whose facets do not match that of a query. In this section, we formally define this task as *facet mismatch classification* and discuss some traditional approaches to address this problem.

5.3.1 Facet mismatch classification

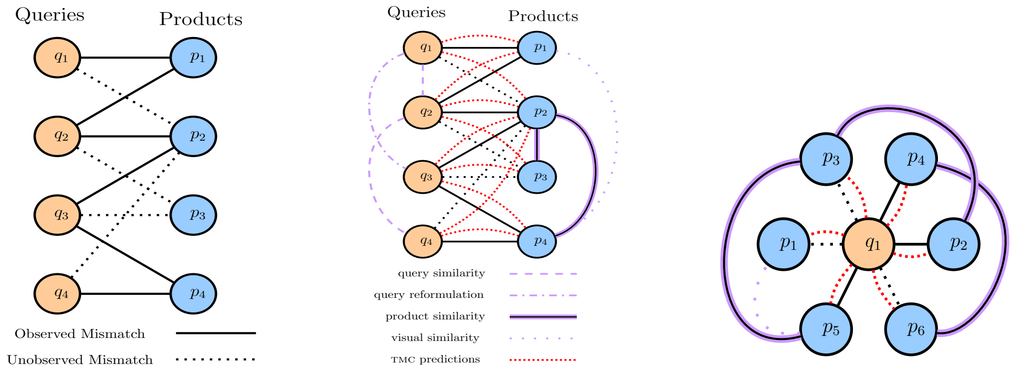
Facet mismatch classification is the general task of classifying a (query, product) pair as matched (or relevant) along one or more facets. Formally, we define facet mismatch classification for a single facet as follows. Note that generalizing this definition for more than one facet is straightforward.

Definition 3 (Facet mismatch classification)**.** *Consider a set of all possible queries $\mathcal{Q}$ and a set of all possible products $\mathcal{P}$. Given a query $q \in \mathcal{Q}$ and a relevance model M such that $M(q) = \mathbf{p}_q$ where $\mathbf{p}_q$ is a ranked list of products returned as relevant to the query q by the model M. Let f be a facet so that $f(q)$ and $f(p_q^i)$ are indicator variables for the facet being present in the query, and the i^{th} product in $\mathbf{p}_q$. Then p_q^i is a facet mismatch if $\gamma_{q,p_q^i} := \nVdash(f(p_q^i) \neq f(q)) = 1$, where $\nVdash$ is an indicator function.*

5.3.2 Traditional approach

The above mentioned problem can be seen as a classification problem with a task of predicting γ_{q,p_q^i} for any given (q, p_q^i) pair. As facet mismatch is a more subtle classification problem than traditional relevance, one cannot simply use user logs and CTR to obtain training data. The data required for training consists of pairs of (q, p_q^i) along with a label γ_{q,p_q^i} where the labels are typically annotated by human curators. Human curation implies that the training datasets are typically much smaller than standard ranking datasets.

Any binary (or multilabel) classifier such as logistic regression or deep neural networks [20] can be used to perform this task, with the caveat that the method must lend itself to fast online predictions. We refer to using such traditional classifiers for performing facet mismatch classification as *traditional mismatch classification* (TMC). To be able to handle low latency, one can perform this classification task using an off-the-shelf industry workhorse model like Gradient Boosted Decision Tree (GBDT) [49, 29].



(a) Facet mismatch classification problem as edge labeling problem.

(b) Structured facet mismatch classification with multiple relational structure.

(c) A micrograph representation for a query.

Figure 5.1: The facet mismatch classification problem as a structure prediction problem. Black dotted edges represent unobserved facet matches. A black solid edge represents an observed facet mismatch and has a value of zero or one. The prediction task is to infer the values for the black dotted edges based on the available structural relationships (all other edges).

In particular, for search and information retrieval systems, GBDTs have been shown to handle both categorical and ordinal features with efficient training and fast real-time inference [69]. The GBDT models trained in search applications use a mix of text and behavioral features, depending on either the query, product or both. A set of joint features ($\varphi(q, p_q^i)$) is used to make prediction on the facet mismatch classification problem:

$$\rho_{q,p_q^i} = GBDT(\varphi(q, p_q^i)) \tag{5.1}$$

In this work we use GBDT models as our TMC. In later sections we show how we make use of this score (ρ_{q,p_q^i}) in our model. TMC uses this score to determine whether there is a facet mismatch, i.e., $\gamma_{q,p_q^i} = \mathbb{1}(\rho_{q,p_q^i} > t)$, where $t \in (0, 1)$ is a threshold.

5.4 Relational structure and micrographs

The facet mismatch classification problem can also be seen as an edge labeling problem on a graph. Consider z queries ($q_1 \dots q_z$) and j products ($p_1 \dots p_j$) as nodes to the left and right side of a bipartite graph (see Fig. 5.1a, here z and j are set to four). The existence of an edge between any query q and product p in this graph represents either textual or behavioral match between a (query, product) pair i.e., $p \in \mathbf{p}_q$. A solid edge between a (query, product) pair indicates an observed mismatch or match (human annotated to one or zero) and dotted edge indicates that the mismatch value need to be inferred. Our goal here is to infer whether an edge is a mismatch or not given a few edge labels. Consider an example query "black apple iphone", all "iphone" products match on the product-type facet and form an edge to the query with value zero. Other product types such as iphone cases and screen protectors have an edge to the query with value one, as they do not match on the product type facet. The label for some edges are known (manually labeled by human judges) and our task is to use the existing edge labels to infer the labels for the unknown edges.

While TMCs can be used to perform facet mismatch classification, they suffer from a major drawback. They assume strong conditional independence in the data (i.e., the value assigned for ρ_{q,p_q^i} is conditionally independent of other pairs in the data given q and p_q^i). This assumption makes inference very scalable. However, it completely ignores the relationship between the query and the list of products for which we need to determine facet mismatch. We depict these additional relationships in Fig. 5.1b. The edge between queries represents many possible relations, such as semantic similarities between queries, query intent relation, and so on. Similarly, the connection between products can be a lexical or semantic similarity and co-purchase behavior. Further, the predictions produced by TMC can also be represented as an edge between (query,product) and can be used as preliminary mismatch scores. The presence of additional edges makes the prediction task γ_{q,p_q^i} dependent on related products and queries. Therefore, a prediction ρ_{q,p_q^i} is no longer conditionally independent and joint predictions have to be performed. For instance, if for some query q, $\gamma_{q,p_q^1} = 1$, and p_q^1 and p_q^2 are connected to each other via a similarity edge, then we'd expect $\gamma_{q,p_q^2} = 1$ This implies that the label assigned for both the edges depend on each other and needs to be predicted jointly. This form of classification is commonly known as *collective classification* [136].

There have been many approaches proposed in varied applications to perform collective classification [136, 4, 82, 112, 165]. The primary issue with such methods is that the amount of time required to perform inference grows rapidly with increases in number of nodes and relations. The heterogeneity of the structural relationships adds to this complexity.

The models that perform collective classification are typically transductive in nature, meaning we need to perform a full (graphical model) inference for a given customer query at run-time. Furthermore, the queries themselves can be arbitrary and precomputing the results and serving them at runtime is not possible. These reasons have precluded the use of such methods for search and information retrieval tasks. We show that by carefully constructing graphs, we can perform inference at runtime. The idea is to break up the graph so that we can perform inference over several smaller (query independent) graphs in parallel. For this reason, we break up the graphs per query and consider (query, product) and (product, product) relationships. We refer to this smaller, more tractable graph as a *micrograph*. Fig. 5.1c shows an illustration. Formally, we define a micrograph as follows:

Definition 4 (micrograph)**.** *A* micrograph *is a graph G, with the vertices being a query q and the list of top-k products $\mathbf{p}_q$ obtained through a retrieval model $M(q)$. The edges in the graph correspond to known (q, p_q^i) labels, and any product-product (p_q^i, p_q^j) edges.*

Typically, a customer does not scroll past a small number of items in response to a query. Hence, we focus our attention to a small k in the above definition, $k \approx 10$. This allows us to use micrographs with very a small number of nodes, making it possible to perform real-time inference.

In this work, we improve the predictions made by the TMC model using these micrographs. We can generate query-product edges as predicted by the TMC for any given (query,product) pair. However, some of these edges may be of low-confidence, or in some cases incorrect. There will also be edges between products, based on co-purchases or semantic similarities which can be computed for all product pairs. The task now is to perform a joint prediction on facet mismatch edges between query and product using all the above mentioned observed data in the micrograph.

Given these micrographs that encode different information in the graph, the next step is to reason over this graph to improve the facet mismatch score. We view this task as performing inference in a graphical model provided by the micrograph. A graphical model created using the micrographs will contain observed random variables as the observed edges in the micrograph and the unobserved random variables to infer are the facet mismatch edge between query and products. In Section 5.5, we elaborate on our SMC and S^{2}MC which combine the micrograph information and perform joint predictions through inference in a graphical model. We use PSL framework to generate the graphical model and perform inference. The PSL framework produces a specific type of graphical model called the *Hinge Loss Markov Random Field* (HL-MRF) [11]. An advantage of HL-MRFs is that we can cast the HL-MRF inference as a convex optimization program and use existing solvers to obtain an exact solution. In the following section, we discuss a brief review of HL-MRF and their relation to PSL.

5.5 Structured Mismatch Classification

We can represent all relationships in a micrograph using PSL and create an HL-MRF to perform efficient inference at run-time. In this section, we define the specific relations that were used to improve facet mismatch classification. We begin by using the prediction scores produced by a TMC for a particular (query, product) edge and propagate this information to other related products. We construct (product, product) edges using the semantic similarity between them. We use latent representations of the products (based on their meta-data) to compute the similarity score. Further, we distinguish high-confidence scores to further improve the overall predictions in the micrograph.

5.5.1 Using TMC Predictions

In the transductive setting, where making a prediction requires an inference step, we need a large set of edges with labeled data to use in a graphical model [45]. However, as explained in the previous sections, obtaining such large amounts of ground truth data is expensive and time-consuming. Instead, one can use the output of an existing discriminative model as the "seed" labels on the edges of the graph. In particular we make use of the scores produced by an underlying TMC, trained on existing ground truth information as labels. The following rules encode the TMC scores:

$$\textsc{tmc}(Q, P) \rightarrow Mismatch(Q, P) \tag{5.2}$$

$$\neg\textsc{tmc}(Q, P) \rightarrow \neg Mismatch(Q, P) \tag{5.3}$$

$Mismatch(Q, P) \in [0, 1]$ is the target predicate to be inferred. $\textsc{tmc}(Q, P) \in [0, 1]$ is the prediction score produced by the TMC for facet mismatch. The above rules incorporate the pairwise classifier which encodes the signal from multiple behavioural and lexical features of the query and product. Note that we can combine scores from multiple classifiers in this way, creating an ensemble of multiple TMCs. We restrict ourselves to a single underlying classifier here for ease of exposition. In our subsequent rules we make use of additional information to improve the predictions.

5.5.2 Using Product Similarities

We can propagate the product scores to similar products to perform joint inference. The primary idea is as follows: if a particular (query, product) pair is a facet mismatch, then substitutable products should also be a facet mismatch for the same query.

There are many ways of computing similarities between products, and an advantage of PSL is that it supports the use of multiple similarity functions. Here, we make use of a latent product representation v_p for a product p, and then use cosine similarity to form the rules. Product representations are created by averaging word embeddings of the title words, the latter of which is learned using word2vec [103]. A similarity predicate can be created using the cosine distance between two vectors, i.e., $Similar(p_1, p_2) = \frac{\langle v_{p_1}, v_{p_2} \rangle}{\|v_{p_1}\| \|v_{p_2}\|}$. Another common method for defining product similarities is via collaborative filtering, where co-purchased or viewed items can be seen to be similar to each other. We also tried rules that make use of collaborative filtering in our model, however we did not see any improvements. The following rules are used to perform the collective inference on the $Mismatch$ predicate:

$$\begin{aligned} &Mismatch(Q, P_1) \wedge Similar(P_1, P_2) \\ &\qquad \rightarrow Mismatch(Q, P_2) \end{aligned} \tag{5.4}$$

$$\begin{aligned} &\neg Mismatch(Q, P_1) \wedge Similar(P_1, P_2) \\ &\qquad \rightarrow \neg Mismatch(Q, P_2) \end{aligned} \tag{5.5}$$

By combining the above rules with rules (5.2) and (5.3) we can generate an HL-MRF which incorporates micrographs to perform joint predictions. We refer to this model as SMC.

5.5.3 Incorporating Confidences into Mismatch Detection

Propagating the right information in the micrograph is key to improving the predictions of the model. Specifically, we want to be able to boost the performance on (query, product) pairs where the TMC cannot confidently predict whether there exists a facet mismatch by propagating information from other (query, product) pairs where the TMC has high confidence. To this end, we introduce two new predicates called $StrongMismatch$ and $Strong$TMC. A $Strong$TMC prediction is one where the TMC score is above (below) a prespecified threshold indicating a match (mismatch). $Strong\text{TMC}(Q, P)$ exists for all (query, product) pair for which the $\text{TMC}(Q, P) > lim_U$ or $\text{TMC}(Q, P) < lim_L$ and $lim_U, lim_L \in [0, 1]$. To use these strong predictors we introduce two more rules:

$$Strong\text{TMC}(Q, P) \rightarrow StrongMismatch(Q, P) \tag{5.6}$$

$$\neg Strong\text{TMC}(Q, P) \rightarrow \neg StrongMismatch(Q, P) \tag{5.7}$$

$StrongMismatch(Q, P)$ is a target predicate to infer. The above rules tell our model to “trust” the TMC when the latter is confident in its predictions.

It is important to note that the above rules are different from the rules in the previous subsection (like (5.4) and (5.5)). Specifically, the former rules incorporate all scores generated by the TMC using behavioural and lexical features. While the rules (5.6) and (5.7) encode an amount of “trust” in the underlying TMC: the query-product pairs for which the confidence in the classification is high can be used as an important signal to incorporate the structure.

The predictions made using TMC are usually good on the products with strong behavioral data associated with them. However, such information is absent on a majority of items, either due to lack of user signals or bad product curation. The primary idea is to improve the predictions on these products by propagating information from similar products where there is strong behavioral data.

Therefore, we propagate only the strong predictions on micrographs that have them, i.e., on a filtered set of queries. We consider $Strong$TMC scores and $StrongMismatch$ for only those queries that contain at least one product with a strong TMC prediction score and one product without a strong TMC score. The mismatch values for other queries are derived directly from TMC scores and are not altered. We refer to this approach of propagating only the high confidence score as *strong* SMC (S^{2}MC).

This targets queries for which there are products whose classification can be improved by propagating scores. This can be encoded using the following rules:

$$\begin{aligned} &StrongMismatch(Q, P_1) \wedge Similar(P_1, P_2) \\ &\qquad \rightarrow StrongMismatch(Q, P_2) \qquad (5.8) \\ &\neg StrongMismatch(Q, P_1) \wedge Similar(P_1, P_2) \\ &\qquad \rightarrow \neg StrongMismatch(Q, P_2) \qquad (5.9) \end{aligned}$$

The above rule states that only a strong prediction from TMC will be propagated to similar products. Further, this information can be propagated to ($Mismatch$) using the following rules:

$$StrongMismatch(Q, P) \rightarrow Mismatch(Q, P) \qquad (5.10)$$

$$\neg StrongMismatch(Q, P) \rightarrow \neg Mismatch(Q, P) \qquad (5.11)$$

Eventually after performing inference if $Mismatch(Q, P) > t$ then the query-product pair is considered a mismatch.

5.5.4 Regularization via Priors

A prior rule is usually used to regularize the values or adjust for class skewness. A negative prior is usually placed on the target predicates $Mismatch$ and $StongMismatch$. This is encoded in the following manner:

$$\neg Mismatch(Q, P) \tag{5.12}$$

The prior acts exactly like the priors in the Bayesian inference literature. The negation in the prior is reasonable: since we expect the majority of (query, product) pairs to be a match. Further, a negative prior can also be seen as a L2 regularizer used in statistical machine learning [11].

For our model, we use all the rules described so far. The weights for each of these rules are learned through grid search. More details about the learned weights are discussed in Section 5.7.2. Further, we use squared hinge-loss potentials for all our rules. To solve the HL-MRF generated we derive and use a quasi-Newton (convex) optimization approach discussed in the next section.

Note that the final output of the model we build will be Mismatch(Q, P). The rules we define are to make sure that the value of Mismatch(Q, P) is accurate, and ideally better than that returned by the baseline TMC.

We round off this section with a representative example to explain the key idea. Consider the query q to be “black apple iphone”, a popular product (p_q^1) to be “black iphone case” which is falsely associated with q, and a new product (p_q^2) to be “golden iphone case”, which is also falsely associated with the query. The objective is to predict γ_{q,p_q^1}, and γ_{q,p_q^2} correctly as a facet mismatch. We train a TMC that makes predictions on the facet mismatch values. Let $\rho_{q,p_q^1} = 1.0$ and $\rho_{q,p_q^2} = 0.5$. In our approach, using rules (5.8) and (5.9), we improve the γ_{q,p_q^2} by propagating ρ_{q,p_q^1} and jointly inferring the values γ_{q,p_q^1} and γ_{q,p_q^2}. This results in $\gamma_{q,p_q^1} = \gamma_{q_1,p_q^2} = 1.0$.

5.6 Scalability

We want our proposed method to perform efficient inference at web scale (i.e., each micrograph inference should take only a few milliseconds or less). Typically, inference in HL-MRFs are solved using ADMM. Although ADMM is scalable and can handle large datasets, it is not fast enough in terms of convergence to meet the stringent latency constraints of an e-commerce website. Of the many optimization methods second-order (Newton) methods are known to have the fastest convergence. However, their per iteration cost increases as the size of data increases. Since we deal with micrographs, for the inference problem in (2.13) each micrograph can be solved independently. This implies that the instantiated model is small and it becomes feasible to use a quasi-Newton method. Specifically, we use the trust-region Newton method (TRON) [62]. In this section, we show that the optimization problem (2.13) is similar to that of (squared) SVMs, which in turn enables us to use solvers based on TRON [47].

To be able to use a quasi-Newton method we need to ensure that our objective is strongly convex, however, (2.13) is not. We thus add an L2 regularizer to the objective. The new objective function can be written as:

$$f(\mathbf{Y}) = \sum_{i=1}^{\iota} w_i \phi_i(\mathbf{Y}, \mathbf{X}) + \lambda ||\mathbf{Y}||_2^2 \tag{5.13}$$

$$\underset{\mathbf{Y}}{\operatorname{argmax}}\, P(\mathbf{Y}|\mathbf{X}) = \underset{\mathbf{Y}}{\operatorname{argmin}}\, f(\mathbf{Y})$$

$$s.t. 0 \leq y \leq 1, \forall y \in \mathbf{Y}$$

$$\phi_i(\mathbf{Y}, \mathbf{X}) = max(l_i(\mathbf{Y}, \mathbf{X}), 0)^2$$

where λ is a hyperparameter. Note that the regularizer effectively replaces the prior rule for the model [11], and so the value for λ can be the same value as the weight of the prior rule (5.12). We can write the term l_i in the potential function ϕ_i as the following:

$$l_i(\mathbf{Y}, \mathbf{X}) = \mathbf{Y}^T \mathbf{z}_i + c_i \tag{5.14}$$

where $\mathbf{z}_i \in \{1, -1\}^n$ is a vector that indicates which unobserved random variables participate in the potential i, n is the total number of unobserved random variables, and $c_i = \mathbf{X}^T \tilde{\mathbf{z}}_i$, where $\tilde{\mathbf{z}}_i \in \{1, -1\}^m$ is a vector that indicates which observed random variables participate in the potential i, m is the total number of observed random variables. The first derivative for the new objective can be written as:

$$\frac{\delta f(\mathbf{Y})}{\delta \mathbf{Y}} = 2\lambda \mathbf{Y} + \sum_{i \in S} 2 w_i \mathbf{z}_i (\mathbf{Y}^T \mathbf{z}_i + c_i) \tag{5.15}$$

where $S := \{i \in 1, 2, \ldots n, \;\; | \mathbf{Y}^T \mathbf{z}_i > -c_i\}$. The (generalized) Hessian is given by,

$$\frac{\delta^2 f(\mathbf{Y})}{\delta \mathbf{Y}^2} = 2\lambda \mathcal{I} + \sum_{i \in S} 2 w_i z_i z_i^T. \tag{5.16}$$

Using the first and second derivative (5.15 and 5.16) we can use quasi-Newton methods to perform inference. This has two distinct advantages over first-order methods like ADMM:

- The number of iterations needed to converge to the optimal value is often orders of magnitude lower.

- Each iteration of ADMM requires solving a set of linear equations to convergence. In contrast, we only take a few steps of conjugate gradient method to obtain an approximate solution. The approximate solution coupled with the second order updates has shown to be highly successful in practice [58]. This significantly reduces the per iteration cost of the second order method.

We make use of the liblinear package [47]. The SVM objective obtained for L2 regularized L2-loss in the primal form is very similar to the Equation 5.14. Specifically, from (5.14) we see that the c_i play the role of a data-specific margin, while the “labels" for each point can be seen to be -1. Concretely, we can rewrite (5.14) in the squared SVM form as,

$$f(Y) = \sum_{i=1}^{\iota} w_i max(0, c_i - (-1)Y^T(z_i))^2 + \lambda\|Y\|^2. \tag{5.17}$$

The only exception is the extra box constraint on $\mathbf{Y}$ which can be easily enforced [92].

5.7 Empirical Evaluation

In this section, we show the power of using micrographs to improve facet mismatch classification. We use multiple sampled datasets to evaluate our approach. First, we show that the TRON-based method we proposed (5.17) is up to 150x faster than the baseline ADMM solver for the PSL, more specifically for facet mismatch classification. We show that the rules we described earlier are indeed useful and the performance of the classifier suffers when we omit these rules. Finally, we compare and contrast various methods and show that our proposed approach significantly outperforms the baseline methods.

5.7.1 Datasets and Models

To evaluate our approach we use three datasets from a popular product listing website where the user issues a query and is returned a ranked list of products. We refer to these datasets as D1, D2, and D3. The datasets correspond to (query, product) pairs shown to the users, with other features obtained from search logs. The most egregious form of facet mismatch is along the product type facet (compared to brand, color, size, etc.) and we focus on that in this chapter. We used human annotators to obtain the ground truth data for all our (query, product) pairs. If a (query, product) pair does not match along the product type facet, the judge marks the pair as mismatched. For example ("iphone x", "iphone x case") is a product type mismatch and ("iphone x", "iphone x refurbished") is a product type match. We have listed the dataset details in Table 5.1. We use the human annotated labels to perform evaluation only, i.e., we consider all variables generated by the $Mismatch$ and the $StrongMismatch$ predicates as unobserved. We use only aggregated and de-identified information for our experiments (i.e., they do not include personally identifying information about individuals in the dataset).

Table 5.1: Details for the three datasets we use. Even though the dataset contain a small number of queries, the query independent nature of our approach enables our results to hold for even larger datasets.

Dataset	Queries	Products
D1	1194	7790
D2	149	866
D3	591	1959

As mentioned earlier, we used TMC as our baseline model. Each training data point is a (query, product) pair and the features included were lexical (text similarity) and behavioral (likelihood of click, add, and purchase). The target (or dependent variable) for this model is binary indicating whether a specific (query, product) pair is a facet mismatch or not. The TMC model was trained on a human annotated dataset with $\sim$

Table 5.2: Different models and rules used to perform evaluation.

Model name	Rules used (Equation number)	Weights for rules
TMC	5.2 and 5.3	1 and 1
STMC	5.2, 5.3, and 5.12	1000, 1000, and 1
SMC	5.2, 5.3, 5.4, 5.5, and 5.12	100, 100, 10, 10 and 1
S^{2}MC	5.2, 5.3, 5.8, 5.9, 5.6, 5.7, 5.10, 5.11, and 5.12	10, 10, 10, 10, 1000, 1000, 100, 100 and 1

$200K$ query-product pairs. We compare this model to the SMC models proposed in this chapter. The first model defined through PSL adds a prior to the TMC score to regularize/smooth the values (using the prior rule), we refer to this model as *smoothed TMC* (STMC). The next PSL model defined uses the micrograph structure defined through product similarities to propagate the TMC scores and perform a joint prediction on mismatches. We refer to this model as SMC. The final model propagates TMC scores only from high confidence edges through product similarities and perform a joint prediction on mismatches, which we refer to as S^{2}MC. We describe the rules used by the models and their weights in Table 5.2.

The weights used here were obtained by performing a grid search over a set of weight options using a training dataset. To compute the similarities between products, we train a word2vec model using about 30M product titles from the catalog of a popular e-commerce website. Each product embedding is the average of the word embeddings in the title (at first glance this might seem naive, but for our specific datasets title contains sufficient information).

We need to quantify the strength of a strong mismatch using an upper and lower limit on the TMC scores. Specifically, assume that the classifier predicts the probability of a (query, product) pair being a match. Then, we need a threshold lim_U so that any score above lim_U is a strong match and any score under a threshold lim_L is a strong mismatch. At the limit point when $lim_U = lim_L$, all TMC scores are classified as

Table 5.3: Number of queries that uses micrograph (Coverage) for different lower and upper limit.

lim_L	lim_U	Coverage
0.01	0.94	0.96
0.02	0.88	0.90
0.03	0.82	0.83
0.04	0.76	0.78
0.05	0.70	0.73
0.06	0.64	0.69
0.07	0.58	0.64
0.08	0.52	0.60
0.09	0.47	0.58
0.10	0.40	0.54
0.11	0.35	0.50
0.12	0.30	0.43
0.13	0.25	0.36
0.14	0.20	0.24
0.15	0.15	0.00

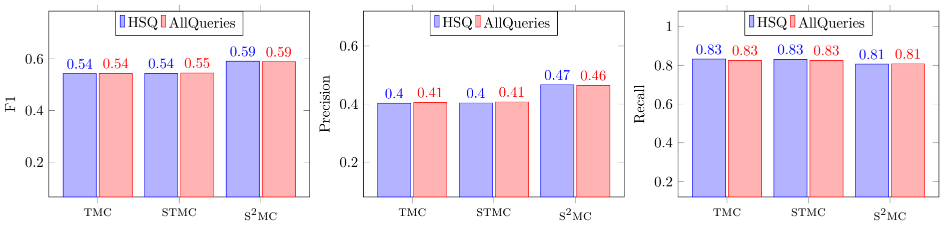


Figure 5.2: Comparison of three models using TMC vs. STMC vs. S^{2}MC on D1 with HSQ coverage 60%.

strong predictions. While this may increase the coverage of queries that have strong predictions, it also decreases the quality of the scores used for propagation. At this limit point, including strong mismatch is the same as the model defined using SMC. Therefore, we choose to use strong predictions for only those queries that contain at least one product with strong prediction and one with weak prediction. The intuition being, a strong prediction can be propagated to a weak prediction if the two products are similar to improve the overall quality of the predictions.

We denote the set of *High Scoring Queries* (HSQ) as queries that are covered by

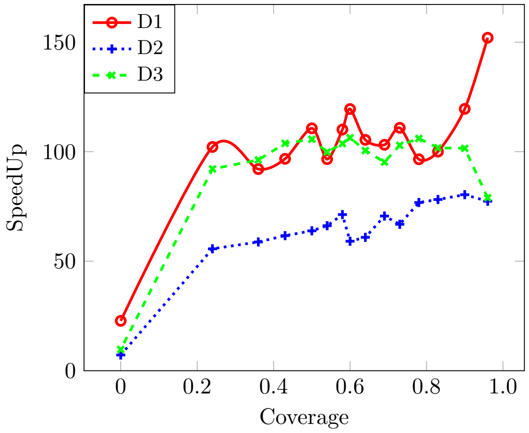


Figure 5.3: Speedup obtained by using TRON over ADMM for performing inference. The speedup increases as coverage increases and we get a speedup of up to 150x on the D1 dataset.

the strong mismatch rules. In the subsequent sections, the suffix 'HSQ' on a model indicates the score obtained by considering only such queries. Lack of a suffix means that we evaluate the performance of the entire dataset. A key thing to note is that the S^{2}MC model uses micrographs for only the HSQs, whereas SMC uses micrograph on all queries.

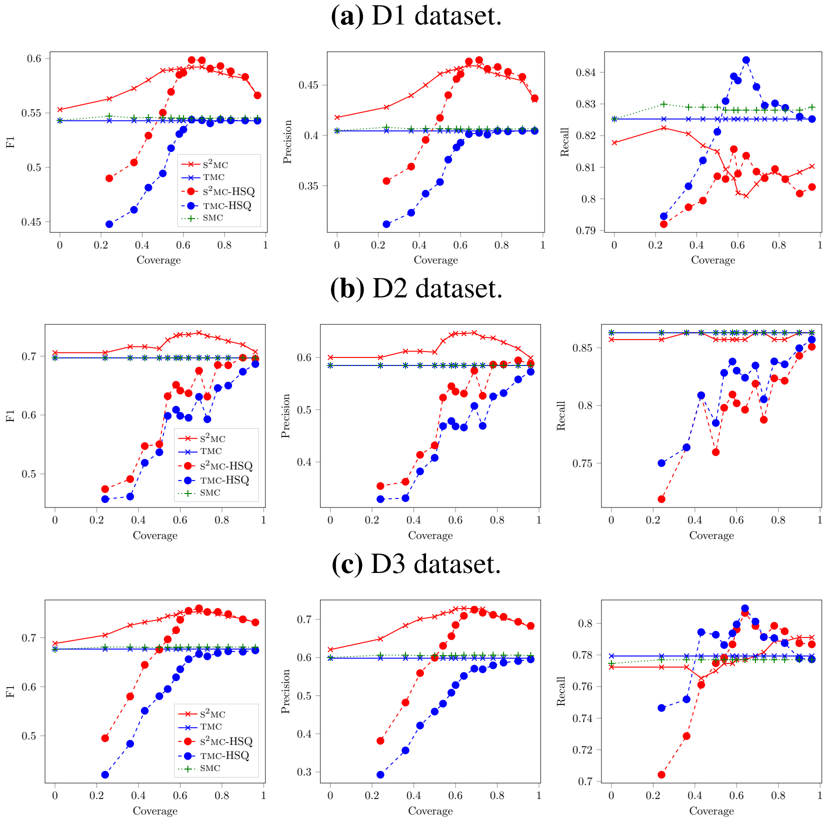
As $lim_L \to lim_U$, the number of HSQs will approach 0. We refer to the fraction of HSQs and total queries as the "coverage", as these will be the queries that make use of micrograph to predict facet mismatch using the S^{2}MC model. We show the different values used for lim_U and lim_L and the corresponding coverage in Table 5.3. Further, we use the PSL open source code[2] to perform inference.

5.7.2 Experimental setup and evaluation

SPEEDUP FROM USING TRON

We show that using TRON makes the S^{2}MC run significantly faster than ADMM. We

[2] http://psl.linqs.org/


(a) D1 dataset.

(b) D2 dataset.

(c) D3 dataset.

Figure 5.4: Precision, recall and F1 for TMC, STMC, SMC, and S^2MC on three different datasets, D1 (top), D2 (middle) and D3 (bottom). We show the metrics for both HSQs and all queries included. Optimal performance is obtained when S^2MC used with threshold of 0.08 and 0.52 is used as lower and upper limit resulting in S^2MC model affecting 60% of queries.

run inference using both TRON and ADMM for all queries in each dataset and report the speedup obtained. To perform inference using TRON, we make minor changes to the open source liblinear package [47] to adapt to the HL-MRF objective. To have a fair comparison, we re-implemented ADMM for HL-MRF in C++.

We report the result of this experiment in Fig. 5.3. As we increase the coverage TRON is up to 150x faster than ADMM. The speedup obtained is minimal when the coverage is 0, i.e., when we used only TMC, and there are no micrographs that slow down ADMM. As we start covering more queries, the speedup also increases. This demonstrates why TRON methods are powerful for online inference when using micrographs, particularly, when we use S^{2}MC. Further, we also observe that when using TRON, per-query predictions time averages to about 0.1 milliseconds.

IMPACT OF MICROGRAPHS

Next, we quantify how much micrographs help, over and above the TMC predictions. Specifically, we consider three cases for dataset D1: using the TMC model, including the prior (rule (5.12)) which we refer STMC, and S^{2}MC. We use the D1 dataset with HSQ coverage 60% [3] (we obtain similar results for datasets D2 and D3). Fig. 5.2 shows the result obtained from this experiment. We observe that when TMC is smoothed with some amount of regularization, there is no improvement in the metrics. However, as we add the information from the micrographs by using S^{2}MC, we see a significant boost. We notice about 7% increase in precision, 6% increase in F1. We also observe similar boost in HSQs. TMC scores tend to be polar for products with high behavioural information. Therefore, the improvement on HSQs indicates that the predictions on products with fewer user interactions (likely tail products) have been improved.

[3] We will show in the sequel that this coverage value performs the best

COMPARISONS WITH MULTIPLE BASELINES

Finally, we compare TMC, STMC, SMC, and S^{2}MC as described in Table 5.2 over multiple coverage values. Note that when coverage is 0, S^{2}MC uses no micrograph and alternatively, when coverage is 1 S^{2}MC uses micrograph for all the queries. We are interested in determining if there is a certain coverage value that maximizes performance. Note that a higher coverage need not necessarily translate to higher performance, since the underlying TMC model might have noisy predictions, leading to noisy edges in the micrographs, in turn leading to incorrect final predictions.

We report our findings in Fig. 5.4. Here we observe that in all datasets there is a clear improvement of precision and F1 in both the HSQ and all queries, while the recall is relatively constant. We observe up to 7% increase in precision for all queries in the D1 dataset and about 6% increase in F1. We see a similar trend in other datasets concerning precision and F1 with a maximum boost of 12% in precision in D2. However, we see a relatively small drop of 2% in recall in the D1 dataset and no drop in recall values in D2 and D3.

We also notice that the evaluation metrics for HSQ in TMC is always lower than the overall value. This is because HSQ contains at least one product for which the TMC prediction score is not high. This implies that for any HSQ q, there is a product p_q^i such that $\gamma_{q,p_q^i} = 1$ but true mismatch label is 0 or vice versa. Therefore, the average number of erroneous TMC predictions per query is higher for HSQs compared to average error for all queries.

In Fig. 5.4 we also observe that the dataset generated using $lim_U = 0.52$ and $lim_L = 0.08$ resulting in 60% coverage yields the maximum improvement in both precision and F1. Another thing to note is that we observe the metric values obtained using SMC are almost the same as that of TMC. Finally, we see that when micrographs are not used at all (coverage = 0), the metric values are almost the same. The slight difference is due to the regularization of the TMC predictions and noise.

5.8 Conclusion and Future Work

We showed in this chapter that structural information can be used to improve facet mismatch classification in modern e-commerce search engines. We introduced the concept of a micrograph, that can be used to incorporate additional structure between queries and products, and reduced the problem to an inference of a graphical model using PSL. The methods we proposed yield impressive gains over baseline methods. We also re-cast the problem with a strongly convex objective, allowing us to use scalable second order approaches and make the inference viable to real time vending of search results. Through experiments, we show that our approach achieves 150X speedup over existing solvers and up to 6% improvement in terms of F1 score.

We hypothesize that incorporating the (query, query) edges in our micrograph will improve the effectiveness of our approach even further. However, keeping in mind the latency constraints of our approach, we have not included them in this chapter. As part of the future work, we would like to explore incorporating this additional information, while not loosing on the latency requirements.

Chapter 6

A Taxonomy of Weight Learning Methods for Statistical Relational Learning

This chapter[1] addresses the issue of weight learning in SRL. Here, I introduce a new taxonomy of weight learning approaches for SRL models based on search strategies. We develop and show that these approaches better addresses the issue of learning the correct set of weights. Further, I show the effectiveness of our approach by comparing our approach for weight learning with other approaches on several realworld datasets.

6.1 Introduction

Learning the weights of the rules is one of the key challenges for templated rule languages such as MLNs and PSL, since the weighted rules interact in complex ways and cannot be optimized independently. Because of their templated nature, the weights, i.e., the parameters of the model, are used in multiple places in the instantiated graphical

[1] Parts of this chapter appeared in *34th AAAI Conference on Artificial Intelligence (2020)*

model, and the context varies depending on the other rules that have been instantiated. In addition, the corresponding probability distribution is not easy to compute; specifically, computing the normalization constant is often intractable. Further, in many applications the final objective is a user-defined evaluation function which can be arbitrary. Optimizing for such arbitrary functions can be even more challenging.

In this chapter, we introduce four search-based approaches for finding the best set of weights for a model (weight configuration) in weighted logic-based SRL frameworks. The key advantage of these approaches are that they directly optimize the chosen domain performance metric and, unlike other approaches, do not require re-derivation of the loss function for each metric. Our proposed approaches are based on black-box optimization methods used in learning hyperparameters in other machine learning approaches [30, 18]. Our first two approaches *random grid search for weight learning* (RGS) and *continuous random search for weight learning* (CRS) are based on simple yet powerful search approaches popularly used in tuning hyperparameters of deep learning models [17]. While RGS is simple and its effectiveness is determined by the human-specified grid, the effectiveness of CRS is determined by the new sampling space we introduce in this work. Our next approach, *Hyperband for weight learning* (HBWL), is based on the Hyperband algorithm [91] that effectively distributes resources to perform efficient random search. Hyperband has been shown to efficiently allocate resources and maximize the search for the best solution. Finally, our fourth approach, *Bayesian optimization for weight learning* (BOWL), is based on Gaussian process regression (GPR) [129] in a Bayesian optimization (BO) [109] framework. BO is an effective approach for optimization of black-box functions [95, 99, 146, 22] and GPR is a non-parametric Bayesian approach that is often used to approximate arbitrary functions. GPRs have been used extensively with a lot of success for hyperparameter tuning in machine learning [142].

In order to perform efficient and effective search of weights using these approaches, we introduce a new parameter search space, which we refer to as *scaled space* (SS), which is an accurate representation of the true weight space. We show that SS is both accurate and complete in representing the weights. Further, we also show that SS also takes into account the impact of model instantiation (also referred to as *grounding*). As sampling from SS is challenging, we introduce an approximation of SS which enables us to efficiently sample weight configurations to perform search-based weight learning. We develop all our search-based approaches for two powerful SRL frameworks, PSL and MLNs. We perform our empirical study on both PSL and MLNs to show the effectiveness of search-based approaches.

Our contributions in this chapter are as follows: 1) we generalize and introduce a new taxonomy of weight learning approaches in SRL frameworks by reformulating the problem of weight learning as a black-box optimization problem and introducing four search-based approaches, referred to as RGS, CRS, HBWL, and BOWL , to perform this optimization; 2) we introduce a new search space called the scaled space (SS) which we show is an accurate representation of the true weight space; 3) we introduce an approximation of SS which generalizes the search-based approaches and simplifies the process of sampling weight configurations in these methods; 4) we show that the search-based approaches are effective at learning weights in both PSL and MLNs and that these approaches outperform likelihood-based approaches on multiple datasets by up to 10%; and 5) finally we show scalability of search-based approaches and perform elaborate set of experiments to show that, of the four approaches, BOWL is robust to initializations, acquisition function (process of choosing next best point, explained in Section 10), and the hyperparameter used in the sampling of weight configurations.

This chapter is organized as follows. First, in Section 6.2, we briefly discuss the related work. Then, in Section 6.3, we provide a brief background on MLNs, PSL, black-box optimization, Bayesian optimization, and Gaussian process regression which are essential in understanding our approaches. Next, along with a motivating example, we introduce our four search-based weight learning approaches for MLNs and PSL in Section 6.4. Then, in Section 6.5, we introduce our novel projection and its approximation which are crucial to the success of our approaches. We then extend our approaches to accommodate for negative weights in Section 6.6, which is essential to fully support MLNs. In Section 6.7, we evaluate all of our approaches on several realworld datasets and metrics. Finally, in Section 6.8 we conclude and discuss potential future work.

6.2 Related Work

Many effective and efficient weight learning approaches have been proposed for SRL frameworks. Most approaches introduced maximize some form of likelihood of the model [96, 14]. Approaches such as [96, 14, 139] perform efficient discriminative learning by approximating the true likelihood with the MAP estimation. Maximizing pseudo-likelihood [115] was shown to be efficient at performing weight learning. Other approaches approximate the likelihood through contrastive divergence [96]. Further approaches such as [113, 13] assume latent variables and use expectation maximization to perform effective learning. There are also large-margin based approaches for learning [63, 152, 66]. Other methods that use SVMs [64, 63, 12] to augment a metric loss to indirectly optimize a user-defined metric. More recently, [133, 35] attempt to make the learning process scalable by performing approximate counting in order to avoid grounding mechanism.

6.3 Background

In this section, we first briefly review required background on black-box optimization and Gaussian process regression (GPR) which serve as the foundation for our proposed approaches.

6.3.1 Black-box optimization

Black-box optimization is a well studied technique, especially in the context of hyperparameter tuning [17, 137].

Definition 5 (Black-box optimization)**.** *Given a black-box function* $\gamma(\tilde{\mathrm{x}}) : \mathbb{R}^d \rightarrow \mathbb{R}$*, where* d *is the input dimension, the task of finding an* $\tilde{\mathrm{x}}$ *that yields the optimal value for* $\gamma(\tilde{\mathrm{x}})$ *in a predefined amount of time is called* black-box optimization.

The goal of black-box optimization is to find the best possible value for $\tilde{\mathrm{x}}$ that optimizes the function $\gamma(\tilde{\mathrm{x}})$ in a predefined amount of resources (generally, number of epochs or time). While this process can be embarrassingly parallel for some approaches, the general strategy can be defined through a sequential setup. The general approach for black-box optimization is to define a search space, choose a point $\tilde{\mathrm{x}}$, evaluate $\gamma(\tilde{\mathrm{x}})$ to update a model, and repeat this process until the resources have been exhausted. Finally, the $\tilde{\mathrm{x}}$ with the best $\gamma(\tilde{\mathrm{x}})$ is returned. The procedure is summarized in Algorithm 2. While the process of selection of $\tilde{\mathrm{x}}$ and evaluation of $\gamma(\tilde{\mathrm{x}})$ on different points can be simple and made to run in parallel for algorithms like RGS, CRS, and HBWL introduced in this chapter (Section 6.4), other algorithms like BOWL (also introduced in this chapter) use BO framework with Gaussian process to approximate γ and assume a serial setup to ensure optimal selection of the next point on which to evaluate.

In the context of BO, various strategies have been proposed to choose the next point to evaluate given the previous evaluations [145, 89, 109, 153]. Each strategy is encoded through an acquisition function α. The objective of these strategies is to minimize the number of epochs required to find the best solution. A simple black-box Bayesian approach iteratively obtains a point to explore from the acquisition function α using the prior distribution; then the function γ is evaluated to obtain a new outcome at that point which is then used to update the posterior. Gaussian process regression (GPR) is a non-parametric Bayesian approach which is effective in performing black-box optimization in a BO framework.

Algorithm 2: Black-box optimization via search

```
Result: $\tilde{\mathbf{x}}^*$ : point with the best value for $\gamma(\cdot)$
$\tilde{\mathbf{X}}$ = search space;
while stopping criteria not met do
    choose a $\tilde{\mathbf{x}} \in \tilde{\mathbf{X}}$;
    update the model of choice with $(\tilde{\mathbf{x}}, \gamma(\tilde{\mathbf{x}}))$;
    update $\tilde{\mathbf{x}}^*$ if current $\gamma(\tilde{\mathbf{x}})$ is better than previous value or based on the
      model
end
```

6.3.2 Gaussian Process Regression

A Gaussian process (GP) is fully characterized by its mean function μ_0 and either a positive definite covariance matrix $\mathbf{K}$ or a kernel function k. Consider a finite set of s inputs $\tilde{\mathbf{X}} = \tilde{\mathbf{x}}_{1:s}$ and a random variable $g_i = \gamma(\tilde{\mathbf{x}}_i)$ representing the function γ evaluated at $\tilde{x_i}$ and let $\tilde{y_i}$ be the noisy output of the function. In GP, we assume that $\mathbf{g} = g_{1:s}$ is jointly Gaussian and $\tilde{y_i}$ given $\mathbf{g}$ is Gaussian. The generative model is of the form: $\mathbf{g}|\tilde{\mathbf{X}} \sim \mathcal{N}(\mathbf{m}, \mathbf{K})$, and $\tilde{\mathbf{y}}|\mathbf{g} \sim \mathcal{N}(\mathbf{g}, \sigma^2\mathbf{I})$, where $m_i = \mu_0(\tilde{\mathbf{x}}_i)$, $\mathbf{K}$ is an $(s \times s)$ positive definite matrix such that $K_{i,j} = k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$. Since the distributions are Gaussian and using the kernalization trick [129], the posterior mean and variance given a set of observed data can be written as:

$$\mu_s(\tilde{\mathbf{x}}_{s+1}) = \mu_0(\tilde{\mathbf{x}}_{s+1}) + k(\tilde{\mathbf{x}}_{s+1})^T(\mathbf{K} + \sigma^2\mathbf{I})^{-1}(y - m)$$
$$\sigma_s(\tilde{\mathbf{x}}_{s+1}) = k(\tilde{\mathbf{x}}_{s+1}, \tilde{\mathbf{x}}_{s+1}) - k(\tilde{\mathbf{x}}_{s+1})^T(\mathbf{K} + \sigma^2\mathbf{I})^{-1}k(\tilde{\mathbf{x}}_{s+1})$$

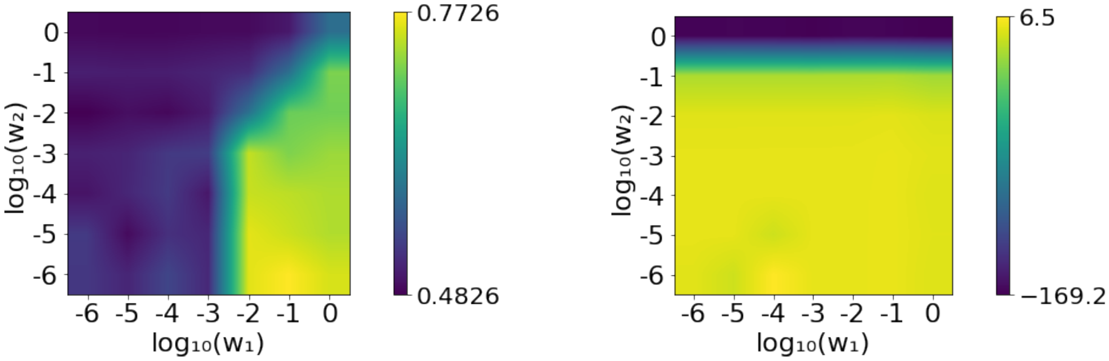
where $k(\tilde{\mathbf{x}}_{s+1})$ is a kernel function applied to the inputs with observed function evaluations and the new input; i.e., it represents the covariance between observed inputs and any unobserved input. Using the above expressions, the mean and variance for any point can be computed. There is a suite of kernel functions available in the literature [129]. Note that the kernel function should be chosen based on the problem domain and it is often the key to finding the best approximation of the true function.

6.4 Search-Based Approaches for Weight Learning

As mentioned earlier, commonly used approaches for rule weight learning in SRL are generally based on maximizing a likelihood function. In this section, we first give a motivating example that highlights the issues with likelihood-based approaches and then we propose four search-based approaches to learn weights in SRL frameworks.

6.4.1 Motivating Example

Consider the model in Example 1 implemented in PSL. Fig. 6.1 shows the performance of the model using PSL as we vary the rule weights logarithmically from 10^{-6} to 1.0. Fig. 6.1 (a) shows AUROC and Fig. 6.1 (b) shows the log-likelihood of the model. Lighter shades (yellow) represent a high value and darker shades (dark blue) represent a low value. We observe that the AUROC is maximized when the first rule's weight is 0.1 and the second rule's weight is 10^{-6}. However, the likelihood is not maximized at these weights. For this model and dataset, we observe that the likelihood is not well



(a) AUROC **(b)** log-likelihood

Figure 6.1: Heat map of AUROC and log-likelihood for the model in Example 1. The lighter color indicates higher values; higher values are desired for both metrics.

correlated with the AUROC. While this behavior is shown using PSL, similar outcome can be seen when using MLN models as well.

6.4.2 Problem definition

Consider a SRL model[2] with r template rules where each rule $i \in \{1 \ldots r\}$ is associated with a weight $w_i \in \mathbb{R}^+$. Grounding all the rules with data D yields a set of m observed random variables $\mathbf{x} = \{x_1, \ldots, x_m\}$, n unobserved random variables $\mathbf{y} = \{y_1, \ldots, y_n\}$, and ι potentials $\phi = \{\phi_1, \phi_2, \ldots, \phi_\iota\}$. The unobserved random variables $\mathbf{y}$ are inferred by optimizing Equation 2.3. Further, all unknown random variables are associated with corresponding ground truth $\mathbf{y}^* = \{y_1^*, \ldots, y_n^*\}$ used to compute evaluation metrics. Let $\mathbf{w} = \{w_1, \ldots, w_r\}$ be the vector representing the set of rule weights, i.e., the weight configuration. Next, let $\omega(\mathbf{y}, \mathbf{y}^*) : (\mathbf{y}, \mathbf{y}^*) \to \mathbb{R}$ be a problem-specific evaluation metric (e.g., accuracy, AUROC, or F-measure) and let $\gamma(\mathbf{w}) : \mathbf{w} \to \omega(\mathbf{y}, \mathbf{y}^*)$ be the same function ω parameterized by $\mathbf{w}$ that maps weights to the metric. Then the objective of weight learning can be expressed as finding the set of weights that maximize the function γ which represents the true metric function ω, i.e.,

[2] In this chapter, we only refer to MLN or PSL programs as SRL models

$\text{argmax}_{\mathbf{w}} \gamma(\mathbf{w})$. The objective of the search-based approaches is to find an approximate function $g \approx \gamma$ by sampling t weight configurations from a set of possible weight configurations $\mathbf{W}$ (the weight space). In order to perform this optimization, we introduce four approaches based on hyperparameter search methods in other areas of machine learning. The first two approaches are based on random grid search and continuous search used in deep learning [17], the next approach is based on Hyperband algorithm used in statistical machine learning [91], and the last one is based on BO with GPR used for hyperparameter tuning in deep learning [142].

6.4.3 Random Grid Search for Weight Learning

A straightforward search-based approach to weight learning is an exhaustive exploration over the set of weight configurations $\mathbf{W}$ generated through a user-specified grid of weights. The user-specified grid to generate weight configurations $\mathbf{W}$ is typically constructed by specifying a finite collection of v values $V = \{V_0, \cdots, V_v\}$ that can be assigned as weights for each of the rules, e.g., $V = \{0.01, 0.1, 1.0\}$. If a model contains r rules, then we can define $\mathbf{W}$ to be the r-ary Cartesian product of V, $\mathbf{W} = V \times \cdots \times V$, defining a grid with the intersections representing different weight configurations. Then, for each configuration $\mathbf{w} \in \tilde{\mathbf{W}}$, $\gamma(\mathbf{w})$ is evaluated after performing MAP inference in the SRL model. Finally, the weight configuration with the highest $\gamma(\mathbf{w})$ is selected.

However, a comprehensive grid search is usually infeasible due to the combinatorial explosion in the size of the grid; if a model contains r rules where each rule can take on one of v possible values, then $|\mathbf{W}| = v^r$. Thus, to make the approach tractable (as mentioned in Algorithm 3), we uniformly draw t unique samples from $\mathbf{W}$ to stay within an established budget of resources; this approach is referred to as *random grid search for weight learning* (RGS). It is important to ensure that the resources are used to evaluate distinct weight configurations, therefore at the time of sampling we ensure that every weight configuration chosen has a possibility of yielding a distinct solution. In Section 6.5, we show how two weight configurations that look distinct can yield the same solution at the time of MAP inference (and hence for the value of γ as well) and how we can identify and avoid wasting resources on such inherently identical weight configurations. Therefore, to ensure uniqueness of weight configurations explored, we keep track of a set $\mathbf{W}_{explored}$ which contains all the weight configurations explored so far and the configurations that are inherently the same as an explored weight configuration and sample new weight configuration from $\{\mathbf{W} - \mathbf{W}_{explored}\}$.

Algorithm 3: Random grid search for weight learning

```
Result: $\mathbf{w}^*$ : weight configuration with best evaluation metric among samples
$\mathbf{W}$ = set of weight configurations from the user-defined weight grid;
$t$ = maximum number of weight configurations to explore;
$\mathbf{W}_{explored}$ = weight configurations explored so far;
for $iter \in \{1, \ldots, t\}$ do
    $\mathbf{w}_{next} = Random(\mathbf{W} - \mathbf{W}_{explored})$;
    perform MAP inference to compute $\gamma(\mathbf{w}_{next})$;
    $\forall \dot{\mathbf{w}} \in \mathbf{W}$ such that $\gamma(\dot{\mathbf{w}}) = \gamma(\mathbf{w}_{next})$, add to $\mathbf{W}_{explored}$ ;
    if $\gamma(\mathbf{w}_{next}) > \gamma(\mathbf{w}^*)$ then
        $\mathbf{w}^* = \mathbf{w}_{next}$;
    end
end
```

6.4.4 Continuous Random Search for Weight Learning

A primary drawback of RGS is the need to define a grid over the space which captures weights that will lead to a good model. While specifying a grid might seem straightforward, several unique properties of the weight space makes the process of specifying the right grid non-trivial (details in Section 6.5). Further, specifying grids can easily result in unexpected biases. For instance, one may be tempted to simply define a grid of evenly spaced points in a unit hypercube. However, this leads to a sampling bias towards configurations with moderate ratios which might not be ideal (more details in Section 6.5.4).

Continuous random search for weight learning (CRS) is similar to RGS in that, rather than exploring the entire space, t weight configurations are chosen for evaluation and the highest performing configuration is returned. The difference is that CRS does not define a discrete grid of weights but samples continuously from the search space. Therefore, it is crucial for the search space to be an accurate representation of the true weight space. In this approach (as mentioned in Algorithm 4), we sample t weight configurations to explore $\mathbf{W}_{explore}$ from a Dirichlet distribution as an approximation of the weight space and finally return the weight configuration with the best value obtained for the γ function. Here, the Dirichlet distribution represents the weight space $\mathbf{W}$. In Section 6.5.4, we discuss in detail on why this is an appropriate choice. For CRS the Dirichlet distribution is parametrized by a r-dimensional hyperparameter $\mathrm{A} \in \mathbb{R}^{+r}$, which can be tuned to obtain the best approximation of the space based on the application and prior knowledge. Note that a Dirichlet distribution can generate positive weights only. This can be restrictive for MLNs which support negative weights. In Section 6.6, we show how the sampling approach is extended to support negative weights for MLNs.

Algorithm 4: Continuous random search for weight learning

```
Result: $\mathbf{w}^*$ : weight configuration with best evaluation metric among samples
$t$ = maximum number of weight configurations to explore;
$\mathbf{W}_{explore} \sim Dirichlet(\mathrm{A})^t$, $t$ distinct samples from a Dirichlet distribution;
for $\mathbf{w}_{next} \in \mathbf{W}_{explore}$ do
    perform MAP inference to compute $\gamma(\mathbf{w}_{next})$;
    if $\gamma(\mathbf{w}_{next}) > \gamma(\mathbf{w}^*)$ then
        $\mathbf{w}^* = \mathbf{w}_{next}$;
    end
end
```

6.4.5 Hyperband for Weight Learning

So far, the search-based methods we have discussed iteratively select a set of t weight configurations to explore, $\mathbf{W}_{explore}$, from a set of weight configurations, $\mathbf{W}$ and then run inference until completion for each $\mathbf{w}_i \in \mathbf{W}_{explore}$ in order to calculate $\gamma(\mathbf{w}_i)$. Then the weight configuration $\mathbf{w}^* = \operatorname{argmax}_{\mathbf{w}_i \in \mathbf{W}_{explore}} \gamma(\mathbf{w}_i)$ is chosen as the model weights. Ideally, in these methods, we want t to be as large as possible, as increasing the number of weight configurations explored (t) can potentially improve $\gamma(\mathbf{w}^*)$ obtained. However, it is generally infeasible to have a large t due to limited resources. In order to maximize our gain with the limited resources, we make use of a practical observation that the weight configurations which initially show a slow rate of improvement during inference will tend to converge to a poor $\gamma(\cdot)$ evaluation. For instance, let us assume two potential weight configurations $\mathbf{w}_1$ and $\mathbf{w}_2$ for a SRL model, if MAP inference for both the weight configurations takes $\hat{t}$ iterations to converge to the final solution and results in $\gamma(\mathbf{w}_1) > \gamma(\mathbf{w}_2)$, then it is highly likely that the $\gamma(\cdot)$ computed by interrupting the MAP inference at $\frac{\hat{t}}{\hat{s}}$ number of iterations, where $\hat{s} > 1$, will still result in $\gamma(\mathbf{w}_1) > \gamma(\mathbf{w}_2)$. Therefore, running MAP inference to convergence for all weight configurations could be wasteful. By early termination of unpromising weight configurations a larger number of configurations can be explored resulting in a better overall solution. This idea has been exploited in other areas of machine learning to tune hyperparameters and is referred to as *Hyperband* [91]. Here, we adapt this approach in the context of weight learning and refer to it as *Hyperband for weight learning* (HBWL).

HBWL allows for more exploration while still operating within a budget (generally time or iterations) through adaptive resource allocation and early-stopping. To understand how the algorithm operates, we will first describe *SuccessiveHalving*, a critical subroutine of HBWL, and then describe how SuccessiveHalving is used in

HBWL. SuccessiveHalving (as mentioned in Algorithm 5) requires two input parameters, namely t the total number of weight configurations we wish to explore and B the total number of iterations for MAP inference. [3] Initially t configurations $\mathbf{W}_{explore} = \{\mathbf{w}_1, \cdots, \mathbf{w}_t\}$ are sampled from the search space $\mathbf{W}$. Then, SuccessiveHalving proceeds in rounds. At the start of each round, a fraction of the budget ($b = \frac{B}{t}$) is allocated to the t weight configurations to perform MAP inference and compute $\gamma'(\cdot, b)$ where $\gamma'(\cdot, b)$ is the evaluation metric computed after b iterations of MAP inference. Finally, the weight configurations are ranked based on $\gamma'(\cdot, b)$, the bottom half of the weight configurations are removed, and $|\mathbf{W}_{explore}|$ is reduced to $\frac{t}{\eta}$ where $\eta > 1$ is the proportion of configurations to be removed (for classic SuccessiveHalving $\eta = 2$). This process is repeated for multiple rounds until only one weight configuration remains which is chosen as the $\mathbf{w}^*$.

The hyperparameters B and t of SuccessiveHalving can trade-off between having a large number of weight configurations with a small amount of resource allocated to each configuration (a.k.a. exploration), or a small number of weight configurations with a large amount of resource allocated to each weight configuration (a.k.a. exploitation). The best trade-off between exploration and exploitation is typically unknown. HBWL (as mentioned in Algorithm 6) extends SuccessiveHalving by trying several possible values for the $\frac{t}{B}$ ratio to choose the best explore exploit trade-off. The possible values for $\frac{t}{B}$ are constructed strategically from the two user provided parameters for HBWL, $\hat{R}$ the maximum amount of resource that can be allocated to a single weight configuration and $\eta \in (1, \infty)$ the proportion of weight configurations to be removed in each round of SuccessiveHalving. Each complete execution of SuccesiveHalving in HBWL is referred to as a bracket. Every bracket is parameterized by the values

[3] Note that, for simplicity, in the algorithm we overload B to be number of iterations in MAP inference in the first round instead of maximum number of iterations. The maximum number of iterations in Algorithm 5 is $B\eta^{log_\eta(t)}$, where η is a parameter defined in HBWL which represents the proportion of weights removed every round in SuccessiveHalving.

(t, B, s) that are constructed uniquely for each bracket using $\hat{R}$ and η, where s is the number of initial configurations being tested in that bracket. HBWL chooses a bracket size of $s = \lfloor \log_\eta(\hat{R}) \rfloor + 1]$ and decrements it every round until one. Finally, the weight configuration with the best value for function γ is returned.

```
Algorithm 5: SuccessiveHalving
  Result: $\mathbf{w}^*$, $\gamma^*$ : weight configuration with the best evaluation metric and the
          metric value $\gamma(\cdot)$
  $B$ = number of iterations in MAP inference for first round;
  $\eta$ = the proportion of weight configurations to be removed in each round;
  $\mathbf{W}_{explore}$ = weight configurations to be evaluated;
  $t = |\mathbf{W}_{explore}|$;
  $b = B$ ;
  $iter = 0$;
  while $t > 1$ do
      $iter$++;
      $\forall \mathbf{w} \in \mathbf{W}_{explore}$ perform MAP inference with max iterations set to $b$;
      if $\mathrm{argmax}_{\mathbf{w} \in \mathbf{W}_{explore}} \gamma'(\mathbf{w}, b) > \gamma^*$ then
          $\gamma^* = \max_{\mathbf{w} \in \mathbf{W}_{explore}} \gamma'(\mathbf{w}, b)$;
          $\mathbf{w}^* = \mathrm{argmax}_{\mathbf{w} \in \mathbf{W}_{explore}} \gamma'(\mathbf{w}, b)$;
      end
      $t = \lfloor \frac{t}{\eta} \rfloor$;
      $b = B\eta^{iter}$;
      $\mathbf{W}_{explore} = top_t(\mathbf{W}_{explore})$, ;
      // $top_t$ ranks $\mathbf{W}_{explore}$ based on $\gamma'$ and returns top $t$ weights;
  end
```

6.4.6 Bayesian Optimization for Weight Learning

Next, we introduce BOWL (Bayesian Optimization for Weight Learning), which uses GPR to perform weight learning in the BO framework.[4] Previously discussed approaches make no assumptions about the search space and the evaluation function (γ). They randomly sample weight configurations from the search space and evaluate the

[4] Note that, in our method we loosely refer to this specific way of using GPR in BO framework as Bayesian optimization.

Algorithm 6: Hyperband for weight learning

```
Result: $\mathbf{w}^*$ : weight configuration with best evaluation metric
$\hat{R}$ = maximum number of resources to be allocated;
$\eta$ = the proportion of weight configurations to be removed in each round;
$s_{max} = \lfloor \log_\eta(\hat{R}) \rfloor$, maximum bracket size;
$\gamma^* = -\infty$, the best $\gamma$ value obtained so far;
for $s \in \{s_{max}, s_{max} - 1, \cdots, 1\}$ do
    $t = \lfloor \frac{(s_{max}+1)\eta^s}{(s+1)} \rfloor$;
    $B = \hat{R}\eta^{-s}$;
    $\mathbf{W}_{explore} \sim Dirichlet(\mathrm{A})^t$, $t$ distinct samples from a Dirichlet distribution;
    $\mathbf{w}_s, \gamma_s = SuccessiveHalving(B, W_{explore})$;
    if $\gamma_s > \gamma^*$ then
        $\gamma^* = \gamma_s$;
        $\mathbf{w}^* = \mathbf{w}_s$;
    end
end
```

function γ. This process can be made more efficient by assuming that the metric obtained by two weight configurations $\mathbf{w}_1$ and $\mathbf{w}_2$ are likely to be similar ($\gamma(\mathbf{w}_1) \approx \gamma(\mathbf{w}_2)$) if the distance between them is small. This implies that when searching the space we can make use of this information and either explore more diverse weight configurations or exploit and choose weight configurations closer to previously best performing configurations. This can be obtained by using BOWL . Next we explain BOWL and our choices for both the kernel function in GPR and the acquisition function in BO which are key in determining the performance of BOWL .

A high-level sketch for BOWL (as mentioned in Algorithm 7) is as follows: first, a weight configuration $\mathbf{w} \in \mathbf{W}$ is chosen using an acquisition function α (discussed in Section 10). Next, inference is performed using the current weight configuration $\mathbf{w}$, and $\gamma(\mathbf{w})$ is computed. Then GPR is updated with $\mathbf{w}$ and $\gamma(\mathbf{w})$. Finally, after t iterations, the weight configuration that resulted in highest value for γ is returned. As mentioned earlier, there are two primary components of BOWL that need to be defined: the kernel function used in GPR and the acquisition function α.

In order to use GPR and choose a kernel function, we must make an assumption about the function γ. Here, we assume that the function γ is smooth. This assumption is true if the problem is well-conditioned and the metric function ω being optimized is a smooth function, such as *mean square error* (MSE). For now, we make this assumption (justified further in Section 10), and choose the *squared exponential kernel* as the kernel for the GP:

$$k(\mathbf{w}_i, \mathbf{w}_j) = \tilde{\sigma} \cdot exp^{\{\frac{\Delta_{i,j}}{2\rho^2}\}} \tag{6.1}$$

where $\tilde{\sigma}$ is the amplitude, ρ is the *characteristic length-scale*, and $\Delta_{i,j}$ is the distance between the two weight configurations $\mathbf{w}_i$ and $\mathbf{w}_j$. ρ and σ are the kernel hyperparameters. The scaling factor ρ affects the smoothness of the approximation (a large value implies more smooth) and the number of iterations required to explore the space. We choose ρ such that a reasonable exploration of the space is possible in t iterations. The value of $\tilde{\sigma}$ is chosen based on the range of the metric being learned.

The distance function Δ is crucial in determining the co-variance between two weight configurations. Ideally, if the distance between two weight configurations is zero then the output of the function γ should be the same. And, as the distance between the two weight configurations increases, the correlation between the output of the γ function should go to zero. In Section 6.5.2 we introduce a new projection for the weights and show that distances measured in the projected space exhibit these properties.

Justification for Squared Exponential Kernel

As mentioned earlier, the squared exponential kernel makes a strong smoothness assumption on the γ function. While this might seem restrictive, we argue that this is a reasonable assumption in the context of weight learning in SRL. To do this, we constrain ourselves to only those metrics ($\omega(\mathbf{y}, \mathbf{y}^*)$) that are smooth with respect to the

Algorithm 7: Bayesian optimization for weight learning

```
Result: $\mathbf{w}^*$ : weight configuration with best evaluation metric among samples
$\mathbf{W} = Dirichlet(A)$;
$t$ = maximum number of weight configurations to explore;
for $iter \in \{1, \ldots, t\}$ do
    $\mathbf{w}_{next} = \text{argmax}_{\mathbf{w} \in \mathbf{W}}\, \alpha(\mathbf{w})$; $\alpha$//an acquisition function chosen from
      Section 10;
    perform MAP inference to compute $\gamma(\mathbf{w}_{next})$;
    update GPR with $\gamma(\mathbf{w}_{next})$;
    if $\gamma(\mathbf{w}_{next}) > \gamma(\mathbf{w}^*)$ then
        $\mathbf{w}^* = \mathbf{w}_{next}$;
    end
end
```

random variables (such as MSE). Note, our assumption is that the function γ is smooth and γ is parametrized with $\mathbf{w}$ and not the random variables $\mathbf{y}$. Hence, it is non-trivial to prove smoothness in γ. With the above constraint on the possible metrics, we know that if a small change in $\mathbf{w}$ leads to a small change in $\mathbf{y}$, then the function γ is also smooth. We formally define smoothness of the function γ as follows:

Definition 6. *Given two sets of weight configurations* $\mathbf{w}_1$ *and* $\mathbf{w}_2$ *for a SRL model with* r *rules that generates* n *unobserved random variables* $\mathbf{y}$*, the function* γ *is considered to be smooth if* $\Delta_{1,2} < \epsilon$ *where* $\epsilon \to 0$*, then* $||\mathbf{y}_1 - \mathbf{y}_2||_2 < \nu$ *where* $\nu \to 0$*,* $\mathbf{y}_1$ *and* $\mathbf{y}_2$ *are the random variables inferred using weights* $\mathbf{w}_1$ *and* $\mathbf{w}_2$ *respectively.*

This directly leads to the conditioning of the problem. If a problem is well-conditioned then our assumption about the smoothness of γ is precise. If the problem is ill-conditioned then this assumption fails to hold and the function learned in BOWL could be a poor approximation of γ. Further, in practice we observe that small changes in weights generally do not affect the γ function significantly which indicates smoothness. Even though we assume $\omega(\mathbf{y}, \mathbf{y}^*)$ is a smooth function such as the MSE to justify squared exponential kernel, in our empirical evaluation we observe this is effective even on other non-smooth evaluation functions such as accuracy.

Acquisition Function

Another crucial component of our algorithm to be defined is the acquisition function α. The function α determines the next weight configuration on which to evaluate the function γ, i.e., $\mathbf{w}_{next} = \mathrm{argmax}_{\mathbf{w} \in \mathbf{W}} \, \alpha(\mathbf{w})$. Since our approach approximates the function γ with g, we would like to choose points that allow us to learn the approximation g while also maximizing the metric γ. To achieve this, we consider four well studied acquisition functions in the context of BO.

Upper confidence bound (UCB) [145]: is an optimistic policy with provable cumulative regret bounds. The acquisition function can be written as:

$$\alpha(\mathbf{W}) = \mu(\mathbf{w}) + \psi \cdot \sigma(\mathbf{w})$$

where μ and σ are the mean and variance predicted by the GP and $\psi \geq 0$ is a hyperparameter set to achieve optimal regret bounds.

Thompson sampling (TS) [153]: is an information-based policy that considers the posterior distribution over the weights $\mathbf{W}$. The acquisition function can be written as:

$$\begin{aligned} \alpha(\mathbf{W}) &= \tilde{p}(\mathbf{w}) \\ \tilde{p}(\mathbf{w}) &\sim \mathcal{N}(\mu(\mathbf{w}), \sigma(\mathbf{w})) \end{aligned}$$

where $\tilde{p}$ are samples obtained from the distribution computed at the point $\mathbf{w}$.

Probability of improvement (PI) [89]: is an improvement-based policy that favors points that are likely to improve an incumbent target τ. The acquisition function can be written as:

$$\alpha(\mathbf{W}) = \mathbb{P}(\gamma(\mathbf{w}) > \tau) = \boldsymbol{\mathcal{F}}\Big(\frac{\mu(\mathbf{w}) - \tau}{\sigma(\mathbf{w})}\Big)$$

where $\boldsymbol{\mathcal{F}}$ is the standard normal cumulative distribution function and τ is set adaptively to the current best observed value for γ.

Expected improvement (EI) [110]: is an improvement-based policy similar to PI. But, instead of probability, it measures the expected amount of improvement. The acquisition function can be written as:

$$\alpha(\mathbf{W}) = \Big\{(\mu(\mathbf{w}) - \tau)\boldsymbol{\mathcal{F}}\Big(\frac{\mu(\mathbf{w}) - \tau}{\sigma(\mathbf{w})}\Big) + (\sigma(\mathbf{w}))\mathcal{F}\Big(\frac{\mu(\mathbf{w}) - \tau}{\sigma(\mathbf{w})}\Big)\Big\}$$

where $\mathcal{F}$ is the probability density function of a standard normal distribution function.

6.4.7 Efficiency of Search-Based Approaches

In practice, it is inefficient to use search-based approaches for high-dimensional problems. For instance, GPR has been shown to work best when the number of dimensions is less than 50 [162]. This makes weight learning an ideal use case for search-based approaches, because typically SRL models have just tens of rules and most often the number of rules does not exceed 50.

The success of all the weight learning approaches discussed so far relies on two important factors: 1) the weight configurations chosen are accurate representation of the true search space; and 2) the distances measured between weight configurations correlate to the solution obtained by using them. In the next section, we first show that the weight space in both PSL and MLN are redundant making the representation imprecise and weight configuration distances do not correlate to the solution obtained. Next, assuming positive weights for rules, we introduce a novel projection that address these challenges. Finally, we also provide an efficient strategy to approximately sample from the projected space ensuring the effectiveness of the search-based approaches introduced.

6.5 Efficient Space to Search for Weights

The weights of a SRL model consisting of r rules is represented via a vector in an r-dimensional space and is referred to as *original space* (OS). However this is an inefficient space to preform weight learning using a search-based approach. This is because there exists many weight configurations which yield the same solution when performing MAP inference in SRL models. The main reason for this is because weights in SRL models are relative and scale invariant at the time of MAP inference. We can show that any SRL model with weights in $\mathbb{R}$ can be re-scaled with a positive constant $\tilde{c}$ without any change to the solution obtained through the objective in Equation 2.3. This shows that weights in SRL models are scale invariant when performing MAP inference.

Theorem 2. *Consider any SRL model with r rules and weights* $\mathbf{w} = \{w_i, \dots, w_r\}, w_i \in \mathbb{R}^+$ *(*$w_i \in \mathbb{R}$ *for MLN). For all weight configurations* $\tilde{c} \cdot \mathbf{w}$ *where* $\tilde{c} > 0$*, the solution obtained for* $\mathbf{y}$ *by performing MAP inference using weights* $\mathbf{w}$ *and* $\tilde{c} \cdot \mathbf{w}$ *are the same, i.e.,* $\operatorname{argmax}_{\mathbf{y}} \hat{s} \cdot \mathbf{E}(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \operatorname{argmax}_{\mathbf{y}} \hat{s} \cdot \mathbf{E}(\mathbf{y}|\mathbf{x}, \tilde{c} \cdot \mathbf{w})$.

Proof. The objective generated by using weights $\tilde{c} \cdot \mathbf{w}$ can be written as:

$$\begin{aligned}
\operatorname*{argmax}_{\mathbf{y}} \hat{s} \cdot \mathbf{E}(\mathbf{y}|\mathbf{x}, \tilde{c} \cdot \mathbf{w}) &= \operatorname*{argmax}_{\mathbf{y}} \hat{s} \cdot \sum_{i}^{\iota} \tilde{c} \cdot w_i \phi_i(\mathbf{x}, \mathbf{y}) \\
&= \operatorname*{argmax}_{\mathbf{y}} \tilde{c} \cdot \hat{s} \cdot \sum_{i}^{\iota} w_i \phi_i(\mathbf{x}, \mathbf{y}) \\
&= \operatorname*{argmax}_{\mathbf{y}} \tilde{c} \cdot \hat{s} \cdot \mathbf{E}(\mathbf{y}|\mathbf{x}, \mathbf{w})
\end{aligned}$$

Since the MAP inference objective using both $\mathbf{E}_{psl}$ and $\mathbf{E}_{mln}$ are scale invariant, the re-scaling of the weight configuration $\mathbf{w}$ to $\tilde{c} \cdot \mathbf{w}$ leaves the solution of the inference unchanged. □

Since in our search-based approaches we optimize w.r.t. a user-defined evaluation metric, and this function depends only on the random variables obtained by MAP inference, we have that the user-defined evaluation metric function is also scale invariant.

6.5.1 Challenges in the Original Space

OS for weights has two fundamental challenges for search-based approaches: 1) OS is redundant and 2) the distance between weights in OS does not translate to true correlation of the solution obtained by using these weights. The redundancy of space is clear from Theorem 2 as the weights on any line intersecting origin in OS will have the same solution. This also means that the Euclidean distance $\delta_{i,j}$ between two weight configurations $\mathbf{w}_i$ and $\mathbf{w}_j$ can be extremely large and still result in the exact same solution and vice versa. The example below clearly illustrates this phenomenon:

Example 6. *Consider a model with two rules* $\mathbf{w} = \{w_1, w_2\}$. *Let us assume three possible weight configurations for this problem:* $\mathbf{w}_1 = \{0.1, 0.1\}$, $\mathbf{w}_2 = \{1.0, 1.0\}$, *and* $\mathbf{w}_3 = \{0.1, 0.0001\}$. *Assuming that the number of groundings for both rules are the same, the weights of the rules in* $\mathbf{w}_1$ *and* $\mathbf{w}_2$ *indicate that both rules are equally important and lie on a line intersecting origin, while in* $\mathbf{w}_3$ *the first rule is* 1000 *times more important than the second rule and is not on the same line. This results in the function* γ *producing the same output for* $\mathbf{w}_1$ *and* $\mathbf{w}_2$, *and potentially a different value for* $\mathbf{w}_3$. *Based on this, the weight configuration* $\mathbf{w}_3$ *should be significantly different from the weight configurations* $\mathbf{w}_1$ *and* $\mathbf{w}_2$, *while* $\mathbf{w}_1$ *and* $\mathbf{w}_2$ *should be similar. Unfortunately, the Euclidean distances measured between the weight configurations,* $\delta_{1,2} = 1.27$, $\delta_{1,3} = 0.09$, *and* $\delta_{2,3} = 1.34$, *do not behave in this manner. The distance* $\delta_{1,2}$ *is much larger than distance* $\delta_{1,3}$. *Therefore, some of the search-based approaches such as* BOWL *would incorrectly infer that the function value of* $\gamma(\mathbf{w}_1)$ *is more correlated with* $\gamma(\mathbf{w}_3)$ *than* $\gamma(\mathbf{w}_2)$. *However, as argued above, we want the opposite behavior.*

In order to address these challenges, we introduce a new projection for the weights.

For the remainder of this section we assume weights of the rules to be positive. While this does not restrict PSL which supports only positive weights, it does constrain MLNs which support negative weights. However, it has been shown that a negative weighted rule in MLNs can be replaced with a negated rule and positive weight with the same magnitude. Further in Section 6.6, we show how we extend and accommodate for positive and negative weights in MLNs when using the search-based approaches.

6.5.2 Scaled Space

In order to perform efficient and effective search, we define a new space for the weight configurations called *scaled space* (SS). SS is a projection of weights onto a relative space. We use the ratio of weights between the rules to define the relative importance of weights in the configuration. This projection eliminates redundancies and results in distances that correspond to the actual correlation between the weight configurations. Formally, we define SS as:

Definition 7. *Given a set of weights* $\mathbf{w} = \{w_1, \ldots, w_r\} \in (0, \infty]^r$*, SS* $\mathcal{E}$ *is a projection defined on* $\mathbf{w}$ *such that* $\mathcal{E}(\mathbf{w}) \in \mathbb{R}^{(r-1)}$ *is given by:*

$$\mathcal{E}(\mathbf{w}) = \{\forall_{i=2}^{r}(ln(w_i) - ln(w_1))\} \tag{6.2}$$

Given the definition of SS, we next define the distance between two weight configurations in SS as:

Definition 8. *The distance* Δ *between two weight configurations* $\mathbf{w}_i$ *and* $\mathbf{w}_j$ *in SS is defined as:*

$$\Delta_{i,j} = ||\mathcal{E}(\mathbf{w}_i) - \mathcal{E}(\mathbf{w}_j)||_2^2 \tag{6.3}$$

Given the definition of SS, we can now show that SS does not have the two challenges mentioned for OS. We first show that any weight configuration on a line intersecting the origin in OS will be represented by the same point in SS eliminating the redundancy that exist in OS. Next we show that in SS ($\mathcal{E}$), a distance of zero ($\Delta_{i,j} = 0$) between two weight configurations $\mathbf{w}_i$ and $\mathbf{w}_j$, implies that the two weight configurations yield the same solution for the random variables $\mathbf{y}$ at the time of MAP inference hence proving accurate representation of distances between weight configurations.

Theorem 3. *Given two weight configurations* $\mathbf{w}_1$ *and* $\mathbf{w}_2$*, if* $\mathbf{w}_1 = c \cdot \mathbf{w}_2$*, i.e.,* $\mathbf{w}_1$ *and* $\mathbf{w}_2$ *lie on a line intersecting the origin in OS then the resultant value in SS will be the same, i.e.,* $\mathcal{E}(\mathbf{w}_1) = \mathcal{E}(\mathbf{w}_2)$

Proof. Given, $\mathbf{w}_1 = c \cdot \mathbf{w}_2$ implies:

$$w_{1,i} = c \cdot w_{2,i}; i \in 1, \ldots, r$$

$\mathcal{E}(\mathbf{w}_1)$ by definition is given by $\{\forall_{i=2}^{r}(ln(w_{1,i}) - ln(w_{1,1}))\}$. By replacing $w_{1,i}$ with $cw_{2,i}$ we get:

$$\mathcal{E}(\mathbf{w}_1) = \{\forall_{i=2}^{r}(ln(w_{2,i}) - ln(w_{2,1}))\} = \mathcal{E}(\mathbf{w}_2)$$

Therefore, if $\mathbf{w}_1 = c \cdot \mathbf{w}_2$ then $\mathcal{E}(\mathbf{w}_1) = \mathcal{E}(\mathbf{w}_2)$. □

Theorem 4. *Given two weight configurations* $\mathbf{w}_1$ *and* $\mathbf{w}_2$*, if* $\mathcal{E}(\mathbf{w}_1) = \mathcal{E}(\mathbf{w}_2)$ *(i.e.,* $\Delta_{1,2} = 0$*) then the solution obtained for* $\mathbf{y}$ *by optimizing Equation 2.3 with both the weight configurations are the same.*

Proof. Let $\mathbf{w}_1 = \{w_{1,1}, \ldots, w_{1,r}\}$, $\mathbf{w}_2 = \{w_{2,1}, \ldots, w_{2,r}\}$ and $\mathcal{E}(\mathbf{w}_1) = \mathcal{E}(\mathbf{w}_2)$. As the two weight configurations are the same in SS, the equality can be written as:

$$\begin{aligned} ln(\mathbf{w}_1) - ln(w_{1,1}) &= ln(\mathbf{w}_2) - ln(w_{2,1}) \\ \mathbf{w}_1 &= \frac{w_{1,1}}{w_{2,1}}\mathbf{w}_2 \end{aligned}$$

Since $w_{1,1} \in (0, 1]$ and $w_{2,1} \in (0, 1]$ are constants, the resulting optimization problems are equivalent:

$$\begin{aligned} \underset{y}{\text{argmax}}\, \hat{s} \cdot \mathbf{E}(\mathbf{y}|\mathbf{x}, \mathbf{w}_1) &= \underset{y}{\text{argmax}}\, \frac{w_{1,1}}{w_{2,1}} \hat{s} \cdot \mathbf{E}(\mathbf{y}|\mathbf{x}, \mathbf{w}_2) \\ &= \underset{y}{\text{argmax}}\, \hat{s} \cdot \mathbf{E}(\mathbf{y}|\mathbf{x}, \mathbf{w}_2) \end{aligned}$$

Therefore, if the distance between two weight configurations is 0 in SS, then the solutions of their corresponding SRL model by optimizing Equation 2.3 are the same. □

Theorem 3 shows that SS is not redundant and Theorem 4 proves the distances in SS are accurate. These theorems show the correctness of SS for weight learning using search-based approaches. Note that in the definition of SS we use the weight of the first rule to compute the projection. This choice is arbitrary and can be switched to any rule without affecting the space.

Example 6. *(Continued) Consider our earlier example. The weights and the distances of our running example in SS $\mathcal{E}$ using Equation 6.2 and 6.3 are: $\mathcal{E}(\mathbf{w}_1) = \{0\}$, $\mathcal{E}(\mathbf{w}_2) = \{0\}$, $\mathcal{E}(\mathbf{w}_3) = \{6.907\}$, $\Delta_{1,2} = 0$, $\Delta_{1,3} = 47.7$, and $\Delta_{2,3} = 47.7$.*

A drawback of SS is that it does not support a weight of zero for any rule in the model. This means that all rules in the configuration must participate in the model. However, in practice, we mitigate this by using SS only to measure distances between weight configurations which is performed by adding a small positive value (e.g., $10^{-\varrho}$, where $\varrho \in Z^+$, $\varrho \gg 0$) to all weights. In order to generate weight configurations for the search, we sample from a hypersphere in OS which has similar properties as SS. We discuss this in detail in the Section 6.5.4.

6.5.3 The Effect of Varied Number of Groundings in the Scaled Space

Our discussion on SS so far has made a very important simplifying assumption, that the number of groundings for each rule in the model is the same. However, the number of groundings produced by different rules are seldom the same and the number of groundings produced by a rule has an impact on the inference of the random variables. The weight associated with each rule is repeated for each ground instance of that rule. This leads to the weight of each rule having varied influence on the minimization of the energy function. For instance, if a model has two equally weighted rules, but one rule produces 10 times more groundings than the other, then that rule implicitly becomes 10 times more important in the model.

Next we show that while the number of groundings has an impact on the solution obtained by SRL models, it does not impact the correctness of SS. We can modify the weights to accommodate the number of groundings of the rules in the model. Consider a model with r rules and let $\beta = \{\beta_1, \ldots, \beta_r\}$ be the number of groundings for each of the r rules. We define a grounding factor κ for each rule. For rule z, the grounding factor $\kappa_z = \frac{\beta_z}{\max(\beta)}$, where $\boldsymbol{\kappa} = \{\kappa_1, \ldots, \kappa_r\}$ is the vector of grounding factors. Therefore, the true weight associated with the z^{th} rule is $\kappa_z \cdot w_z$ and the grounding adjusted weight configuration can be represented as an element-wise dot product between $\boldsymbol{\kappa}$ and w, i.e., $\tilde{\mathrm{w}} = \boldsymbol{\kappa} \cdot \mathrm{w}$. The distance between two weight configurations i and j in OS can be re-written as $||\tilde{\mathbf{w}}_i - \tilde{\mathbf{w}}_j||_2^2$. Similarly the distance in SS can be re-written as $||\mathcal{E}(\tilde{\mathbf{w}}_i) - \mathcal{E}(\tilde{\mathbf{w}}_j)||_2^2$. However, the scaling factor $\boldsymbol{\kappa}$ does not affect the distance in SS as $\boldsymbol{\kappa}$ is constant for both weight configurations and cancels when computing the distance leaving the distance in SS unchanged.

Theorem 5. *Given two weight configurations $\mathbf{w}_i$ and $\mathbf{w}_j$, a set of grounding factors of $\boldsymbol{\kappa}$, and grounding adjusted weight configurations $\tilde{\mathbf{w}_i} = \boldsymbol{\kappa} \cdot \mathbf{w}_i$ and $\tilde{\mathbf{w}_j} = \boldsymbol{\kappa} \cdot \mathbf{w}_j$, the distance measured between both $(\mathbf{w}_i, \mathbf{w}_j)$ and $(\tilde{\mathbf{w}_i}, \tilde{\mathbf{w}_j})$ in SS are equal, i.e. $||\mathcal{E}(\mathbf{w}_i) - \mathcal{E}(\mathbf{w}_j)||_2^2 = ||\mathcal{E}(\tilde{\mathbf{w}_i}) - \mathcal{E}(\tilde{\mathbf{w}_j})||_2^2$.*

Proof. To prove the above theorem we consider the difference between the weight configurations $\mathcal{E}(\tilde{\mathbf{w}_i}) - \mathcal{E}(\tilde{\mathbf{w}_j})$:

$$\begin{aligned}\mathcal{E}(\tilde{\mathbf{w}_i}) - \mathcal{E}(\tilde{\mathbf{w}_j}) &= (ln(\boldsymbol{\kappa} \cdot \mathbf{w}_i) - ln(\kappa_1 \cdot w_{i,1})) - \\ &(ln(\boldsymbol{\kappa} \cdot \mathbf{w}_j) - ln(\kappa_1 \cdot w_{j,1})) \\ &= (ln(\mathbf{w}_i) - ln(w_{i,1})) - \\ &(ln(\mathbf{w}_j) - ln(w_{j,1})) \\ &= \mathcal{E}(\mathbf{w}_i) - \mathcal{E}(\mathbf{w}_j)\end{aligned}$$

Since $\mathcal{E}(\tilde{\mathbf{w}_i}) - \mathcal{E}(\tilde{\mathbf{w}_j}) = \mathcal{E}(\mathbf{w}_i) - \mathcal{E}(\mathbf{w}_j)$, the distances are also equal. □

Theorem 5 shows that the distance measured between two weight configurations in SS is robust while considering the size of their groundings.

6.5.4 Sampling Weight Configurations for Search

Most search-based approaches work by choosing different possible weight configurations to explore and search for the weight configuration with the best evaluation score. In order to do this effectively, we must ensure that the samples generated for exploration are representative of the space. While it might be ideal to directly sample uniformly from SS, this is challenging due to the one-to-many mapping between SS and OS and, as mentioned earlier, points in SS cannot represent rules with zero weights. One straightforward approach to handle this is to uniformly sample weight configurations from the positive quadrant of a unit hypercube in OS and and project the points on to SS to measure distances. The approach can be summarized as:

$$\mathbf{w} \sim \mathit{Unif}([0,1])^r$$

where $Unif$ generates uniform random numbers between $[0,1]^r$. Since, the influence of weights in SRL models are scale invariant for MAP inference, we can ensure all possible weight configurations that can be represented in the $\mathbb{R}^{+r}$ can be represented in $[0,1]^r$. However the main problem with this approach is that the resulting configurations will have a low spread in SS as OS has a lot of redundancies and the projection will place several weight configurations around the same region. This is not desirable as we would prefer to have a uniform/large spread of possible weight configurations in SS.

As mentioned earlier, MAP inference in SRL models are scale invariant and hence weights along a line passing through 0 in OS are equivalent, i.e., the direction of a vector starting at the origin in OS is sufficient to represent all weight configurations in SRL models. This implies that the weight configurations obtained by sampling from the surface of an r-dimensional unit hypersphere in the positive quadrant represents all possible weight configurations in OS. Therefore, uniformly sampling from the surface of this hypersphere (we only refer to the positive quadrant of the unit hypersphere) is a close approximation of SS.

To show that this is a reasonable space to sample weights from we need to show: first, the surface of the hypersphere is complete and non-redundant and second, if two weight configurations in OS are represented by the same weight configurations on the hypersphere then the solution obtained for $\mathbf{y}$ by both the weight configurations are the same. Every weight configuration given in OS can be easily projected on to the hypersphere in the following way:

$$\mathcal{H}(\mathbf{w}) = \frac{\mathbf{w}}{||\mathbf{w}||_2} \tag{6.4}$$

where $||\mathbf{w}||_2$ is the L-2 norm of the vector.

Theorem 6. *Every weight configuration* $\mathbf{w}$ *in OS has an equivalent weight configuration* $\tilde{\mathbf{w}}$ *on the hypersphere such that* $\operatorname{argmax}_{\mathbf{y}} \hat{s} \cdot \mathbf{E}(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \operatorname{argmax}_{\mathbf{y}} \hat{s} \cdot \mathbf{E}(\mathbf{y}|\mathbf{x}, \tilde{\mathbf{w}})$.

Proof. From Theorem 2 we know that weights are scale invariant when performing MAP inference and this implies that the magnitude of the vector generated by a weight configuration does not affect the solution obtained when performing inference in PSL and MLNs. This implies that all r-dimensional unit vectors in the positive quadrant is sufficient to represent all possible weight configuration for MAP inference in PSL and MLNs. All unit vectors can be represented using the surface of a hypersphere. Therefore, surface of an r-dimensional hypersphere removes redundancies of OS and is complete. □

Theorem 7. *Given two weight configurations* $\mathbf{w}_1$ *and* $\mathbf{w}_2$ *such that the projection* $\mathcal{H}$ *of the weight configurations are equal, i.e.,* $\mathcal{H}(\mathbf{w}_1) = \mathcal{H}(\mathbf{w}_2)$*, then the optimization solution obtained for* $\mathbf{y}$ *by performing inference is the same.*

Proof. This is easy to show as the definition of the projection defined in Equation 6.4 is very similar to the projection defined in Theorem 2. The weights are simply rescaled and therefore, by definition and from Theorem 2, it is clear to see that if $\mathcal{H}(\mathbf{w}_1) = \mathcal{H}(\mathbf{w}_2)$ then $\operatorname{argmax}_{\mathbf{y}} \hat{s} \cdot \mathbf{E}(\mathbf{y}|\mathbf{x}, \mathbf{w}_1) = \operatorname{argmax}_{\mathbf{y}} \hat{s} \cdot \mathbf{E}(\mathbf{y}|\mathbf{x}, \mathbf{w}_2)$. □

This shows that the projection on to the surface of a unit hypersphere has properties similar to SS. However, these two are not entirely equivalent as the grounding factor κ plays an important role in the actual impact of weights at the time of inference.

While SS is ideal and distances between weights are preserved even after adjusting for grounding, the distances are not preserved after adjusting for grounding.

Theorem 8. *Given two weight configurations* $\mathbf{w}_1$ *and* $\mathbf{w}_2$ *and their grounding adjusted weights* $\kappa \cdot \mathbf{w}_1$ *and* $\kappa \cdot \mathbf{w}_2$ *(an element-wise dot), the distance between the two weights before and after grounding adjustment are not the same, i.e.,* $||\mathbf{w}_1 - \mathbf{w}_2|| \neq \kappa \cdot \mathbf{w}_1 - \kappa \cdot \mathbf{w}_2$.

Therefore, we can conclude that while the surface of a hypersphere does not have all the properties of SS it is a reasonable approximation of SS and weights can be samples from the hypersphere.

Therefore, we treat the hypersphere as an approximation of SS and sample weight configurations from the hypersphere but compute distances in SS. Uniform samples of weight configurations from the surface of the hypersphere can be obtained by first sampling points from a standard multivariate-normal distribution and projecting the values to the hypersphere [111, 98] (since we want samples only from the positive quadrant we project all samples on to this quadrant by taking the absolute value):

$$\mathbf{s} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$
$$\mathbf{w} = |\frac{\mathbf{s}}{||\mathbf{s}||}|$$

$\mathbf{0}$ is a r-dimensional zero vector and $\mathbf{I}$ is an r-dimensional identity matrix.

While uniform sampling from the surface of a hypersphere ensures that every orientation of the weight vector is equiprobable, in practice this might not always be desirable. The primary reason for this is that for any weight configuration sampled from the hypersphere, if we choose two weights $w_i < w_j$ the $P(\frac{w_i}{w_j} < 0.1) \approx 0.11$ (assuming $r = 2$). This implies that the ratio between weights will typically be close to one which is not ideal as we would expect the evaluation metric to show large variance with larger ratios. In order to circumvent this, we propose another sampling strategy which gives us full control over the distribution of the weight configurations. We sample from an r-dimensional Dirichlet distribution which generates samples from the probability simplex. It is easy to see a one-to-one correspondence of the probability simplex and the surface of a sphere. The hyperparameter of the Dirichlet distribution can be modified to generate samples skewed towards the center (all equal weights) or the poles (extreme ratios) or anywhere in between. The sample generation process is as follows:

$$\mathbf{w} \sim Dirichlet(\mathrm{A}) \tag{6.5}$$

where $\mathrm{A} \in R^{+r}$ is the hyperparameter that defines the Dirichlet distribution.

While sampling from a hypersphere with different densities can be tricky one easy way to do this is to sample from a Dirichlet distribution which samples from the probability simplex and project the values on to the sphere. The Dirichlet distribution accepts the hyperparameter $\mathrm{A} \in \mathbb{R}^{+r}$ which controls the spread of the distribution. It is easy to see that every point on a probability simplex can be uniquely projected on to a hypershpere. Here in Fig. 6.2 we visualize the distribution generated by using different values of A for a model with three rules. We vary the value of alpha form all 10s to all 0.001 and plot the projection on to a sphere which is in OS on the left and SS (which is 2-dimensional space) on the right. We can observe that as the value of A is reduced the samples in OS are getting concentrated to the poles and in SS we see the samples spread wider and choosing larger ratios. We can choose this value based on our application. For instance when the task is to perform rule pruning, it is more desirable for the ratio of weights to be more extreme and hence choose a small value for A and if the task is to fine tuning weights without making large changes, then a higher value of A might be more suitable.

6.6 Accommodating Negative Weights in Markov Logic Networks

In this section we discuss how the sampling strategy discussed in Section 6.5.4 is modified for MLNs to accommodate negative weights when using CRS, HBWL, and BOWL . Since the Dirichlet distribution samples from the probability simplex, the weights sampled are strictly non-negative. To introduce the possibility of negative weights, we first sample weight configurations from the Dirichlet distribution and then randomly select an orthant in the r-dimensional Euclidian space. This random selection of an orthant has the same effect as independently flipping the sign of each weight in the sampled configuration so every orthant is equiprobable. We further apply the positive scale invariance property for weight configurations to ensure the uniqueness of samples. While no sampling is needed for RGS in MLNs, we ensure uniqueness of explored configurations by projecting the values on to a unit hypersphere instead of SS. In order to use BOWL in MLNs, instead of computing distance in SS, we compute the Euclidean distances of the weight configurations by projecting the weights onto the hypersphere. [5]

6.7 Empirical Evaluation

In this section, we evaluate the search-based approaches for weight learning on various realworld datasets. We investigate four research questions through our experiments:

[Q1] How do search-based approaches perform on realworld datasets compared to the existing methods?

[5] Note this may lead to over generalization of the function as two points might be very close to each other but have significantly different outcome on the γ function (as mentioned in Example 6).

[Q2] Which search-based approach performs better weight learning in SRL?

[Q3] Are search-based approaches scalable?

[Q4] Are search-based approaches robust?

In order to answer these questions we selected five realworld datasets from different domains for which SRL models have promising results [11, 84].[6] Details of these datasets are as follows:

Jester: contains 2,000 users and 100 jokes [57]. The task is to predict user's preference to jokes.

LastFM: contains 1,892 users and 17,632 artists. The task is to recommend artists to users by predicting the ratings for user-artist pairs.

Citeseer: contains 2,708 scientific documents, seven categories, and 5,429 directed citations. The task is to assign a category to each document.

Cora: is similar to Citeseer dataset, but contains 3,312 documents, six categories and 4,591 directed citations.

Epinions: contains 2,000 users and 8,675 directed links which are positive and negative trust links between users. The task is to predict the trust relation.

We evaluate the three search-based methods (RGS, CRS, and BOWL) for both MLNs and PSL, and HBWL for PSL only. While the implementation of RGS, CRS, and BOWL are efficient to run for both MLNs and PSL (more details in Section 6.7.2), HBWL with MLNs on our datasets is not as efficient and each experiment was estimated to take about a month. While the search-based approaches have been fully integrated into PSL codebase, for MLNs, we implement search-based methods as wrappers, i.e., we use an external script to generate weights and use Tuffy as a black-box evaluator. Because of this, in HBWL, the SuccessiveHalving step requires us to reground the same model multiple times which substantially increases time required to run. We

[6] Models, code, and data: https://github.com/linqs/srinivasan-mlj20

use MLE, MPLE, and LME as baseline methods for PSL and use DN implemented in Tuffy as our baseline for MLN. As MLNs are discrete, we perform evaluations only on the three discrete datasets, Citeseer, Cora, and Epinions. For each dataset, depending on the problem, we leverage the metric that has been used for the problem to measure its performance. Hence, we report *MSE* and *AUROC* for the Jester and LastFM datasets, *categorical accuracy (CA)* and *F1* for the Cora and the Citeseer datasets, and *AUROC* and *F1* for the Epinions dataset. Further, all our experiments were run on a machine with 16 cores and 64GB of memory.

6.7.1 Performance analysis

To address [Q1] and [Q2], we compare the performance of all search-based approaches RGS, CRS, HBWL, and BOWL with MLE, MPLE, and LME in PSL and DN in MLN on several metrics. For the datasets, we use the 8 folds generated by [11] for Citeseer, Cora, Epinions and Jester and 5 folds generated by [84] for LastFM and perform cross validation. We perform a paired t-test (p-value ≤ 0.05) across methods to measure statistical significance. In RGS for PSL, we specify the set $V = \{0.001, 0.01, 0.1, 1, 10\}$, while for MLNs we use $V \cup -V$. For CRS, HBWL, and BOWL , we specify each dimension of A to be 0.05 for PSL (more experiments with different values can be found in Section 6.7.3) and 0.1 for MLNs. In HBWL for PSL, $\hat{R}$ represents the number of iterations of ADMM inference and is set to 25000 and η is set to 4. For RGS, CRS, HBWL, and BOWL , the maximum number of weight configurations to explore in order to approximate the user-defined evaluation metric function is set to $t = 50$. Although in our experiments we observed that the best metric value is usually obtained at $t < 25$ (especially for BOWL , this is likely because the function we intend to learn has several flat regions). We also use a stopping criterion for BOWL which terminates the exploration if the standard deviation at all sampled

Table 6.1: Performance of PSL weight learning methods across datasets. The best scoring methods (with $p < 0.05$) are shown in bold. Note: the metric values are rounded to three points of precision, making some numbers the same, but the significance tests were performed on values with six point precision.

Method (Metric)	MLE	MPLE	LME	RGS	CRS	HBWL	BOWL
Jester (MSE)	0.053	0.068	0.063	0.053	0.053	**0.052**	**0.053**
Jester (AUROC)	0.730	0.700	0.702	**0.762**	0.744	**0.756**	**0.761**
LastFM (MSE)	0.061	**0.060**	0.061	**0.061**	0.062	0.064	0.063
LastFM (AUROC)	0.548	0.552	0.549	0.574	0.572	0.556	**0.587**
Citeseer (CA)	0.835	0.837	0.839	**0.844**	**0.844**	**0.845**	**0.844**
Citeseer (F1)	**0.283**	0.293	**0.303**	**0.321**	**0.322**	**0.321**	**0.319**
Cora (CA)	**0.881**	**0.888**	**0.889**	**0.888**	**0.887**	**0.888**	**0.889**
Cora (F1)	**0.404**	**0.439**	**0.442**	**0.438**	**0.438**	**0.438**	**0.443**
Epinions (AUROC)	**0.811**	**0.792**	**0.807**	**0.814**	**0.797**	**0.810**	**0.780**
Epinions (F1)	**0.712**	**0.711**	**0.712**	**0.711**	**0.712**	**0.711**	**0.713**

weight configurations is less than 0.5. MLE, MPLE, LME, and DN are allowed to run for 100 iterations or until convergence whichever is smaller. For BOWL , we use UCB as our acquisition function with $\psi = 1$ to favor modest exploration. However in Section 6.7.3 we show that similar performance can be obtained with PSL by using the other acquisition functions discussed in Section 10. Other hyperparameters that we use for BOWL are: $\tilde{\sigma} = 0.5, \rho = 1$, and the mean function is a constant 0.5. We set the value of ρ to one after exploring different values in $[10^5, 10^{-5}]$, and we set $\tilde{\sigma}$ to 0.5 as our metrics are mostly in the range $[0, 1]$.

Table 6.1 and 6.2 show the comparison between search-based approaches and other methods across the different datasets for both PSL and MLN respectively. In each row of the table, the best performing method and those that are not significantly different from the best performing method are shown in bold. We first analyze the performance results of PSL in Table 6.1. Here we observe that the search-based approaches are the best performing method across all the datasets and metrics. Specifically, we observe that BOWL is the best or not significantly different from the best performing method on all datasets and metrics (except LastFM MSE). For the Epinions and Cora datasets, we observe that there is no statistically significant difference among all approaches on both metrics. However, on the Citeseer dataset, we observe that, for CA, all search-based approaches perform better than other approaches. An interesting observation here is that on Citeseer dataset the weights found by MLE and LME perform as well as search-based approaches on F1 but not on CA. This further strengthens our motivation to directly optimize for the evaluation metric rather than the likelihood. In LastFM, we observe that BOWL is significantly better than all other approaches on AUROC, however, RGS outperforms BOWL in the MSE metric on LastFM. One reason for likelihood-based approaches to perform better in LastFM MSE could be because the rating values mentioned in LastFM was constructed by fitting a negative binomial distribution [84]. This could potentially imply that maximizing the likelihood in PSL could lead to minimizing the MSE. Finally on the Jester dataset, HBWL and BOWL perform the best on both metrics. Overall, as mentioned earlier, search-based approaches perform better than other approaches and in general even the worst performing search-based method is better than likelihood-based approaches (like in Jester and LastFM AUROC). These experiments clearly show that search-based approaches are better suited for weight learning in PSL. Further, of the search-based weight learning approaches we observe that BOWL performs the best followed by HBWL. This is because BOWL performs smart exploration of space to approximate the evaluation function and HBWL evaluates more points by smart resource allocation to get better approximation compared to CRS and RGS.

Next, in Table 6.2 we analyze the performance of the MLN weight learning methods on all datasets and metrics. We again see that search-based methods achieved the best performance for every dataset and metric. Except for F1 score in the Cora dataset, where DN performs as well as search-based approaches, DN performs worse than search-based approaches on all datasets and metrics. This observation further supports the use

Table 6.2: Performance of MLN weight learning methods across datasets. The best scoring methods (with $p < 0.05$) are shown in bold.

Method (Metric)	DN	RGS	CRS	BOWL
Citeseer (CA)	0.829	0.828	0.829	**0.833**
Citeseer (F1)	0.267	0.251	0.253	**0.281**
Cora (CA)	0.867	0.865	0.866	**0.871**
Cora (F1)	**0.339**	**0.333**	**0.342**	**0.355**
Epinions (AUROC)	0.614	**0.689**	**0.699**	0.659
Epinions (F1)	**0.705**	**0.708**	**0.711**	**0.710**

of search-based weight learning methods across SRL frameworks. Similar to the PSL experiments, we observe that overall BOWL performs the best followed by HBWL. BOWL is either the best method and or not significantly different from the best method on all but the Epinions dataset when evaluating the AUROC metric. The reason for poor performance of BOWL in Epinions with AUROC could be because Epinions has the maximum number of rules (20 rules) compared to all other datasets. Further, since distances for BOWL in MLN are computed in OS, the function approximated for AUROC might need more fine tuning which might not be possible in OS leading to poor exploration. On the same dataset for F1 we observe that BOWL performs as well as CRS which is the best. This indicates that the approximation of the evaluation function through BOWL is dependent on the function being approximated as well.

6.7.2 Scalability

In this section, we compare the runtimes of MLE, MPLE, LME, RGS, CRS, HBWL, and BOWL in PSL to measure the scalability of search-based approaches and address [Q3]. We do not compare runtimes of our experiments with Tuffy. While DN is fully integrated in Tuffy, our search-based approaches were implemented as a wrapper to Tuffy. Hence, the runtime numbers of different approaches will not be a fair comparison.[7] The number of parameters to learn in PSL and MLN is equal to the number of rules in the model and the data size translates to the number of groundings generated by the model. In Fig. 6.3a, we show the number of groundings generated by each of the datasets.We also show the number of rules in each model. The Jester dataset produces the largest number of groundings ($\sim 1M$) using seven rules and the Epinions dataset produces the least number of groundings ($\sim 14K$) using the largest model (20 rules).

In Fig. 6.3b, we show the average runtimes measured accross all folds for all approaches on all datasets. Runtimes for some approaches depend on number of groundings while some others depend on number of rules. The runtime for MPLE primarily depends on number of groundings and we observe that as the number of groundings increases the runtime also increases by a factor of ~ 45 from Epinions to Jester dataset. The time taken to run LME depends not only on the number of groundings, but also the complexity of finding the margin. Therefore, we observe that the runtime of LME on the LastFM dataset is higher than the Jester dataset. MLE, RGS, CRS, HBWL, and BOWL depend on number of groundings through the time taken to perform inference on larger models. Since inference in PSL is efficient due to its convex objective, these approaches can scale better with the number of groundings compared to the other approaches. Further, inference time in PSL depends on both the number of groundings (which affects per iteration cost in solving) and ease of solving the optimization (which affects the number of iterations required to converge). This can be observed when we compare runtimes of the above mentioned five methods on the Epinions dataset and Cora dataset. The time taken to run Epinions is two times greater for MLE even though the number of groundings of Epinions is three times smaller. Overall we observe that MLE has the largest increase in runtime (~ 150 fold increase) from Cora to

[7] Even though we reground the model every iteration for search-based methods in Tuffy, we observed that the search-based approaches were at least two times faster than DN in Citeseer and Cora datasets and due to early-stopping, BOWL in MLN was at least 5 times than DN.

Jester dataset. The runtimes of search-based approaches also depend on the number of evaluations (or resources allocated) they are allowed. For a good approximation of the evaluation function the number of evaluation points required increases exponentially with number of rules in the model. Since we fix the maximum number of iterations to 50 for all models, it does not affect our approaches. Specifically, we observe that the runtimes of BOWL across datasets does not vary as much as other approaches. BOWL has the smallest increase in runtime ($\sim$3 fold) between the Cora and Jester datasets across all methods. This is because BOWL has a constant overhead for performing updates in the GP and retrieving the best next set of weight configurations. This also ensures bad weights that converge poorly are chosen less often (as they are likely to yield poor evaluation metric value), hence making it efficient and scalable with number of groundings. Overall we observe that search-based approaches can scale with a large number of groundings and produce the best results on the evaluation metric function.

6.7.3 Robustness

To address [Q4], we ran three sets of experiments: the first experiment is to check how robust search-based methods are w.r.t. different initialization, the second is to evaluate the robustness of CRS, HBWL, and BOWL to the hyperparameter A from the Dirichlet distribution, and the third experiment is to test the effects of choosing an acquisition function on the performance of BOWL .

Varied Initialization

For the first experiment, we perform weight learning with all four search-based approaches using 30 random initializations and report the mean and standard deviation (std) of a metric per dataset in Table 6.3. Note that for this experiment we use UCB as our acquisition function in BOWL . Further, we use only one fold (of the eight folds) per dataset as we intend to measure the variance introduced by different initialization. In Table 6.3, we observe that the standard deviation is very small for Jester dataset and on other datasets except LastFM we observe that the standard deviation introduced by different folds is much higher than initializations indicating robustness to initialization. Finally, on the LastFM dataset, on all approaches except HBWL, we observe a standard deviation larger than the standard deviation across folds obtained in Table 6.1. This is likely because HBWL explores more weight configurations using smart resource allocation. We can conclude that while the search-based approaches are reasonably robust to initialization, this can depend on the dataset and the number of weight configurations they are allowed to try.

Table 6.3: Mean (std) of the metrics obtained by running search-based approaches with varied initialization. We observe that the performance of BOWL is least affected by both initialization.

Datasets	Varied initializations			
	RGS	CRS	HBWL	BOWL
Jester (MSE)	0.053 (0.001)	0.053 (0.001)	0.052 (0.001)	0.053 (0.001)
LastFM (MSE)	0.067 (0.008)	0.067 (0.006)	0.065 (0.002)	0.067 (0.006)
Citeseer (F1)	0.319 (0.007)	0.321 (0.007)	0.322 (0.008)	0.320 (0.007)
Cora (F1)	0.412 (0.006)	0.409 (0.005)	0.411 (0.005)	0.411 (0.005)
Epinions (F1)	0.714 (0.002)	0.713 (0.002)	0.714 (0.002)	0.716 (0.001)

Impact of Hyperparameter A

Here we evaluate the robustness of CRS, HBWL, and BOWL to the hyperparameter A in the Dirichlet distribution when sampling for the weight configurations. We chose four different values for $\mathrm{A} = \{10, 1, 0.1, 0.01\}$ and evaluated on one discrete (Citeseer) dataset and one continuous (Jester) dataset. For Citeseer, we use CA as the evalaution metric and we use MSE for the Jester dataset. We evaluate the three approaches, CRS, HBWL, and BOWL , that are impacted by A in PSL. Table 6.4 shows the metrics obtained for different values of A on both datasets and all methods. Here, we observe that the effect of A on the Citeseer datasest is minimal in all methods. This is likely because the CA function w.r.t. weights is reasonably flat and small changes in weights have minimal impact. Therefore as long as there is at least one sampled point in a region, it is sufficient to get an optimal value and thus all approaches seem robust. Next, when we consider the Jester dataset, we observe that the parameter A has a large impact on CRS followed by HBWL. For a value of $\mathrm{A} = 0.1$ CRS and HBWL perform the best and as it is increased to 10, we observe that both approaches produce the worst MSE value. This is because the space generated by the Dirichlet distribution using these parameter values is not representative of the true weight space for these two approaches. The impact of A is smaller on HBWL than CRS as HBWL explores more weight configurations via smart exploration. Finally, we observe that BOWL is robust to the parameter A as it uses GP to choose the next point.

Table 6.4: Performance of different search-based approaches by varying the hyperparameter A in the Dirichlet distribution. The best metric values in every row is shown in bold.

Datasets	**Methods**	$\mathrm{A} = 10$	$\mathrm{A} = 1$	$\mathrm{A} = 0.1$	$\mathrm{A} = 0.01$
Citeseer (CA)	CRS	**0.844**	**0.844**	**0.843**	**0.844**
	HBWL	**0.843**	**0.844**	**0.843**	**0.843**
	BOWL	**0.844**	**0.844**	**0.845**	**0.843**
Jester (MSE)	CRS	0.064	0.054	**0.053**	0.056
	HBWL	0.061	0.053	**0.052**	0.054
	BOWL	**0.053**	**0.053**	**0.053**	**0.053**

Impact of Acquisition function in BOWL

Our third experiment measures the robustness of BOWL to different acquisition functions. In Table 6.5 we compare the performance of BOWL using four different acquisition functions (UCB, TS, PI, and EI) for all five datasets in PSL (one metric per dataset). On all datasets and metrics, we observe that BOWL is relatively robust to acquisition function and performs similarly for all.

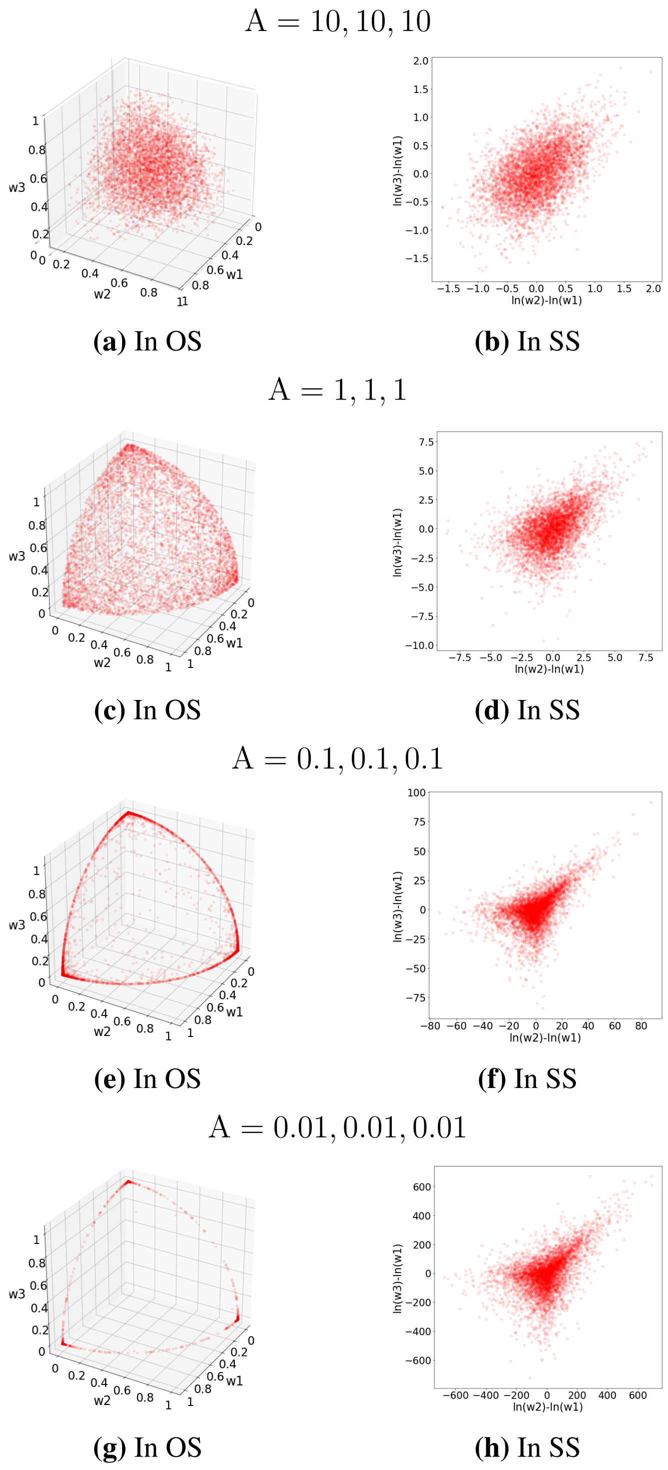
Table 6.5: Effect of metrics obtained by using different acquisition function with BOWL . We observe that the performance of BOWL is unaffected by both acquisition function.

Datasets	Different acquisition functions			
	UCB	TS	PI	EI
Jester (MSE)	0.053	0.055	0.053	0.053
LastFM (MSE)	0.065	0.065	0.066	0.067
Citeseer (F1)	0.324	0.323	0.323	0.323
Cora (F1)	0.440	0.437	0.442	0.439
Epinions (F1)	0.711	0.712	0.713	0.713

Finally, based on these experiments it can be observed that BOWL is the most robust to initialization, hyperparameter, and acquisition function and produces the best evaluation metric.

6.8 Conclusion and Future work

In this chapter, we introduce four search-based approaches to learn weights in SRL. We introduce a novel projection which results in efficient search over the best weight configuration that maximizes a user-defined evaluation metric. We show that search-based approaches improved performance across several metrics on a variety of different realworld datasets. There are many avenues for expanding our work. In this chapter we evaluate the new weight learning approaches on PSL and MLN, as two well-known SRL frameworks. Exploring the effect of search-based approaches on the performance of other SRL frameworks remains as an open path to explore in the future. To perform weight learning using search-based approaches, the SRL model needs to be fully grounded. There are a variety of approaches for avoiding full grounding [132, 133] that would be interesting to integrate into our approach. Further, performance of GPs are highly dependent on the kernel function used. Therefore, in one of our search-based approaches (*Bayesian optimization for weight learning (*BOWL *)*) that use GP, an exploration of different kernels for BOWL could further improve the performance of our method.



(a) In OS

(b) In SS

(c) In OS

(d) In SS

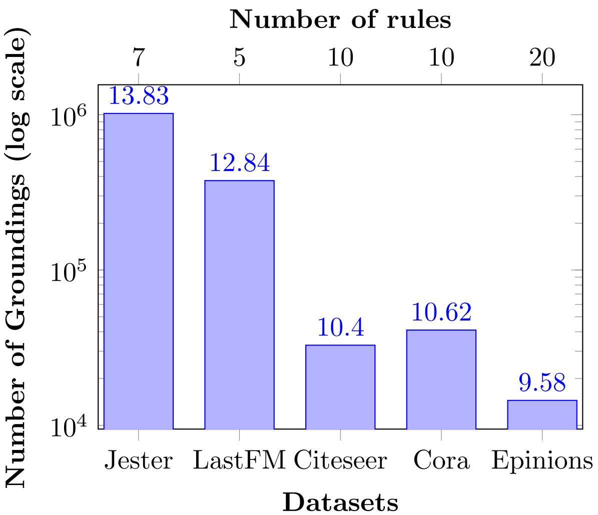
(e) In OS

(f) In SS

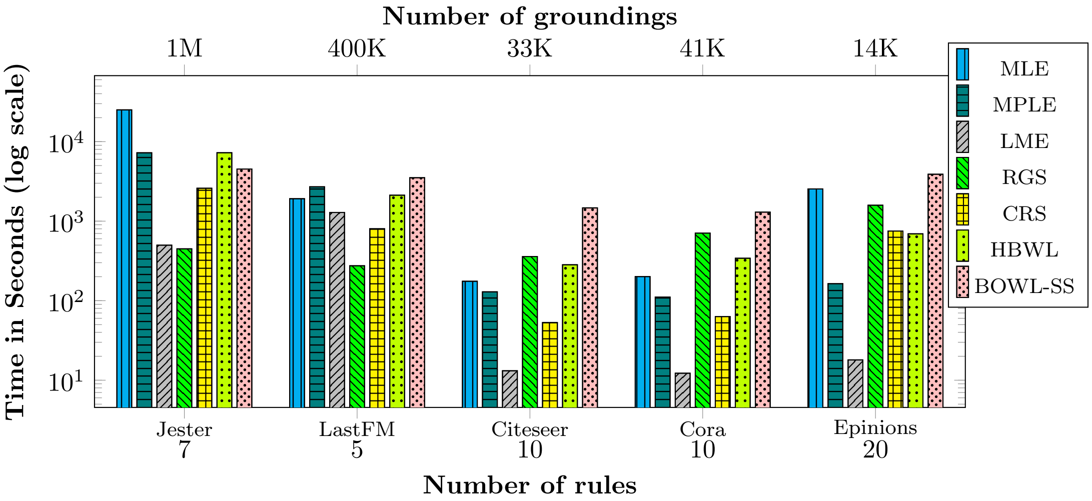
(g) In OS

(h) In SS

Figure 6.2: Visualization of Dirichlet distribution with different values of hyperparameter A for a model with three rules. Visualization shown both in OS and SS.



(a) Groundings generated by different datasets.



(b) Time to learn vs. # of rules and groundings in datasets.

Figure 6.3: Analyzing the scalability of different approaches w.r.t. the number of rules and groundings. When the number of iterations is fixed, search-based approaches scale better with both the number of rules and groundings.

Chapter 7

Conclusion and Future Work

In this dissertation, I addressed several challenges that improve the scalability and accuracy of models generated using probabilistic soft logic (PSL). Specifically, I addressed four fundamental challenges in PSL: 1) how to scale structured prediction using large graphical models; 2) how to perform structured prediction with limited memory; 3) how to perform real-time structured prediction; and 4) how to learn weights in SRL that can maximize arbitrary evaluation functions. To tackle these challenges, I proposed three novel approaches for scaling inference and taxonomy of weight learning approaches that can optimize arbitrary evaluation functions.

7.1 Summary of Contributions

I first introduced a new method to scale MAP inference in graphical models generated through PSL that exploits symmetries in the graphical model referred to as lifted hinge-loss Markov random fields (LHL-MRF). I showed that this approach eliminates symmetries and perform inference on a smaller graph to recover the exact solution obtained by performing inference on the full model. Through extensive empirical evaluation, I showed that this approach can have a significant impact on realworld datasets.

Further, I also performed an analysis to understand the impact of the approach on inference and runtimes on different scenarios.

Next, I addressed the challenge of performing inference in large graphical models on a machine with limited memory by proposing a novel approach called tandem inference (TI). This approach interleaves model instantiation and inference by performing them both in tandem, eliminating the need for large memory. Further, by disk space to stream data and eliminating the need to keep models in memory, the approach scales based on disk space, which is significantly larger than the main memory. Evaluation of several datasets showed that TI can run consistently with limited memory and run faster than traditional approaches making them great for all large datasets.

Next, to address the challenge of performing real-time structured prediction, I introduced an approach based on the trust region Newton method (TRON) for efficient inference on small datasets. To show the power of this approach, I introduced a new collective classification task in a product retrieval domain. I also proposed a structured prediction approach that could outperform traditional methods in this task. Combined with the TRON-based inference engine, I showed that it is possible to perform the structured prediction task in realtime. Through evaluations on multiple realworld datasets, I showed that this structured prediction approach, combined with TRON-based inference, can improve precision by 12% and perform inference 150 times faster than alternating directions method of multipliers.

Finally, to learn rule weights in SRL that directly optimize for evaluation metrics, I proposed a new taxonomy of approaches based on search strategies in black-box optimizations. I introduced a novel and accurate projection space for the weights crucial in learning the approximations of the evaluation functions. I proposed four powerful weight learning strategies that can learn any arbitrary evaluation function, such as the F1 score, using the projection space. Through empirical evaluation with five realworld datasets and two metrics each on two SRL frameworks, I showed that search-based weight learning approaches could significantly outperform likelihood-based approaches. I also analyzed the scalability and robustness of these approaches and showed that some of the search-based methods are scalable and robust to multiple hyperparameters.

7.2 Future Work

In this dissertation, I focused on addressing the challenges of three types of scalability and parameter estimation in graphical models generated through SRL models. While I primarily focused on using PSL as our SRL framework, the approaches introduced in this dissertation are generic. Several of these approaches can be extended to several other SRL frameworks to improve their scalability and accuracy. TI in specific is a powerful technique that can improve the scalability of many other graphical models by eliminating the need to have a large main memory. While this dissertation applies the weight learning approaches to MLNs, a much more involved integration can extend or introduce a new projection space that is more expressive for MLNs. While we showed that search-based approaches are robust to hyperparameters in PSL, we observed this to not be true for MLNs as the projection space introduced for PSL doesn't fully apply for MLNs. Generalizing the projection space can help improve parameter estimation for other SRL frameworks.

In LHL-MRF, we observed that the improvement depended on not just the reduction in model size but also the hardness of the problem generated. While in general, it is common to expect a smaller graphical model to have lower runtimes, in some cases, this is not true. The number of iterations required for ADMM to converge can increase when we eliminate symmetries. This can significantly increase the runtime of the approach, making LHL-MRF ineffective. Three factors affect the hardness of a problem in PSL inference: 1) the structure of the problem induced from rules; 2) structure of the problem induced from data; and 3) weights used in PSL. Analyzing and understanding how each of these factors affects inference in PSL can significantly help address the scalability challenge in PSL.

Next, to address the challenge of memory-constrained optimization, I introduced TI which makes use of the disk to stream the model to perform inference. While this is an effective approach, it is inherently serial, and the gradient-based approach proposed can be sensitive to hyperparameters like learning rate. On the other hand, ADMM-based inference is generally more robust to hyperparameters and is inherently parallel. Combining multiple inference engines such that one can exploit the gradient-based approach's streaming nature and robustness and parallelism of the ADMM-based method could result in a significantly more scalable and efficient mechanism.

While TI and LHL-MRFs address two different types of challenges, they are orthogonal approaches that can be combined to improve the system's scalability significantly. There have been several attempts at performing lifted inference in SRL, and since most of the are bottom-up approaches, they fail to work when model size is too large. TI on the other hand ignores the existence of symmetries and naively assumes no symmetries in the model. By combining these two powerful approaches, we can minimize disk access in TI (the most expensive operation) and perform more efficient inference.

A fundamental challenge this dissertation does not address is structure learning in graphical models generated through SRL frameworks. We improved the accuracy of a model by optimizing the weights of the rules. However, these rules might not be the optimal set of rules one can use with a dataset. We refer to the task of identifying relevant rules from data as structure learning. Many previously proposed approaches perform structure learning in SRL frameworks [102, 80, 81, 126]; however, they generally do not scale to large datasets. Some scalable systems however, limit themselves to only path-based rules [51, 52] or non-collective rules [73, 74]. These generally limit the models learned. By exploiting many of the techniques mentioned in the dissertation, creating a scalable structure learning system could significantly improve the model accuracy.

Another growing area of research in machine learning is interpretability and explainability [1]. With the prolific use of machine learning across applications, it becomes critical to have the ability to explain predictions [44, 2]. Many statistical learning methods are both not very interpretable and explainable. Generally, they need several post-processing steps to be able to interpret the model and generate explanations. However, SRL approaches are uniquely placed as they define a model through weighted logical rules which are inherently interpretable which makes generating explanations more straightforward and accurate [76, 94, 88, 87]. While generating explanations using SRL models can be simpler than statistical methods, they still pose several of the scaling challenges that exist in inference. It will be crucial to ensure that we can generate explanations on large graphical models with high accuracy.

Bibliography

[1] ACM Conference on Fairness, Accountability, and Transparency (ACM FAccT). `https://facctconference.org/`.

[2] A. Adadi and M. Berrada. Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE Access*, 6:52138–52160, 2018.

[3] Somak Aditya, Yezhou Yang, and Chitta Baral. Explicit reasoning over end-to-end neural architectures for visual question answering. In *AAAI*, 2018.

[4] Charu C. Aggarwal. *Data Classification: Algorithms and Applications*. Chapman & Hall/CRC, 2014.

[5] Sanjay Agrawal, Kaushik Chakrabarti, Surajit Chaudhuri, Venkatesh Ganti, Arnd Christian Konig, and Dong Xin. Exploiting web search engines to search structured databases. In *WWW*, 2009.

[6] Babak Ahmadi, Kristian Kersting, Martin Mladenov, and Sriraam Natarajan. Exploiting symmetries for scaling loopy belief propagation and relational training. *MLJ*, 92:91–132, 2013.

[7] Babak Ahmadi, Kristian Kersting, and Scott Sanner. Multi-evidence lifted message passing, with application to pagerank and the kalman filter. In *IJCAI*, 2011.

[8] Duhai Alshukaili, Alvaro A. A. Fernandes, and Norman W. Paton. Structuring linked data search results using probabilistic soft logic. In *ISWC*, 2016.

[9] Eriq Augustine and Lise Getoor. A comparison of bottom-up approaches to grounding for templated Markov random fields. In *SysML*, 2018.

[10] Eriq Augustine, Theodoros Rekatsinas, and Lise Getoor. Tractable probabilistic reasoning through effective grounding. In *ICML workshop on Tractable Probabilistic Modeling*, 2019.

[11] Stephen H. Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. Hinge-loss Markov random fields and probabilistic soft logic. *Journal of Machine Learning Research*, 18:109:1–109:67, 2017.

[12] Stephen H. Bach, Bert Huang, and Lise Getoor. Large-margin structured learning for link ranking. In *NIPS Workshop on Frontiers of Network Analysis: Methods, Models, and Applications*, 2013.

[13] Stephen H. Bach, Bert Huang, and Lise Getoor. Learning latent groups with hinge-loss Markov random fields. In *ICML Workshop on Inferning: Interactions between Inference and Learning*, 2013.

[14] Stephen H. Bach, Bert Huang, Ben London, and Lise Getoor. Hinge-loss Markov Random Fields: Convex Inference for Structured Prediction. In *UAI*, 2013.

[15] Gükhan H. Bakir, Thomas Hofmann, Bernhard Schölkopf, Alexander J. Smola, Ben Taskar, and S. V. N. Vishwanathan. *Predicting Structured Data (Neural Information Processing)*. The MIT Press, 2007.

[16] Islam Beltagy, Katrin Erk, and Raymond J. Mooney. Probabilistic soft logic for semantic textual similarity. In *ACL*, 2014.

[17] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *JMLR*, 13(null):281–305, 2012.

[18] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *NIPS*, 2011.

[19] Indrajit Bhattacharya and Lise Getoor. Collective entity resolution in relational data. *TKDD*, 1:1–36, 2007.

[20] Christopher M Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, 2006.

[21] Stephen P. Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 2011.

[22] Eric Brochu, Tyson Brochu, and Nando de Freitas. A Bayesian Interactive Optimization Approach to Procedural Animation Design. In *SIGGRAPH*, 2010.

[23] Guy Van den Broeck and Adnan Darwiche. On the complexity and approximation of binary evidence in lifted inference. In *NIPS*, 2013.

[24] Hung Hai Bui, Tuyen N. Huynh, and Sebastian Riedel. Automorphism groups of graphical models and lifted variational inference. In *UAI*, 2013.

[25] Hung Hai Bui, Tuyen N. Huynh, and David Sontag. Lifted tree-reweighted variational inference. In *UAI*, 2014.

[26] Michael J. Cafarella, Michele Banko, and Oren Etzioni. Relational web search. In *WWW*, 2006.

[27] Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artif. Intell.*, 172(6-7):772–799, 2008.

[28] Po-Ta Chen, Feng Chen, and Zhen Qian. Road traffic congestion monitoring in social media with hinge-loss Markov random fields. In *ICDM*, 2014.

[29] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *KDD*, 2016.

[30] Marc Claesen and Bart De Moor. Hyperparameter search in machine learning. *arXiv*, 2015.

[31] Paolo Codenotti, Hadi Katebi, Karem A. Sakallah, and Igor L. Markov. Conflict analysis and branching heuristics in the search for graph automorphisms. In *ICTAI*, 2013.

[32] Michael Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *EMNLP*, 2002.

[33] Mayukh Das, Devendra Singh Dhami, Gautam Kunapuli, Kristian Kersting, and Sriraam Natarajan. Fast relational probabilistic inference and learning: Approximate counting via hypergraphs. In *AAAI*, 2019.

[34] Mayukh Das, Yuqing Wu, Tushar Khot, Kristian Kersting, and Sriraam Natarajan. Scaling lifted probabilistic inference and learning via graph databases. In *SDM*, 2016.

[35] Mayukh Das, Yuqing Wu, Tushar Khot, Kristian Kersting, and Sriraam Natarajan. Scaling lifted probabilistic inference and learning via graph databases. In *SDM*, 2016.

[36] L. De Raedt, K. Kersting, S. Natarajan, and D. Poole. *Statistical Relational Artificial Intelligence: Logic, Probability, and Computation*. Morgan & Claypool, 2016.

[37] Luc De Raedt and Kristian Kersting. Statistical relational learning. In *Encyclopedia of Machine Learning*. 2011.

[38] Rodrigo de Salvo Braz, Eyal Amir, and Dan Roth. Lifted first-order probabilistic inference. In *IJCAI*, 2005.

[39] Rina Dechter and Robert Mateescu. AND/OR search spaces for graphical models. *Artif. Intell.*, 171(2-3):73–106, 2007.

[40] Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Lifted relax, compensate and then recover: From approximate to exact lifted probabilistic inference. In *UAI*, 2012.

[41] Guy Van den Broeck, Nima Taghipour, Wannes Meert, Jesse Davis, and Luc De Raedt. Lifted probabilistic inference by first-order knowledge compilation. In *IJCAI*, 2011.

[42] Lingjia Deng and Janyce Wiebe. Joint prediction for entity/event-level sentiment analysis using probabilistic soft logic models. In *EMNLP*, 2015.

[43] Javid Ebrahimi, Dejing Dou, and Daniel Lowd. Weakly supervised tweet stance classification by relational bootstrapping. In *EMNLP*, 2016.

[44] Hugo Jair Escalante, Sergio Escalera, Isabelle Guyon, Xavier Baro, Yagmur Gucluturk, Umut Guclu, and Marcel van Gerven. *Explainable and Interpretable Models in Computer Vision and Machine Learning*. Springer, 2018.

[45] Dhivya Eswaran, Stephan Günnemann, Christos Faloutsos, Disha Makhija, and Mohit Kumar. Zoobp: Belief propagation for heterogeneous networks. *VLDB*, 2017.

[46] Shobeir Fakhraei, Bert Huang, Louiqa Raschid, and Lise Getoor. Network-based drug-target interaction prediction with probabilistic soft logic. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 2014.

[47] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *JMLR*, 9:1871–1874, 2008.

[48] Sholeh Forouzan and Alexander Ihler. Linear approximation to admm for map inference. In *JMLR W&CP*, pages 48–61, 2013.

[49] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 2000.

[50] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. Prentice Hall Press, 2008.

[51] Matt Gardner, Partha Pratim Talukdar, Bryan Kisiel, and Tom Mitchell. Improving learning and inference in a large knowledge-base using latent syntactic cues. 2013.

[52] Matt Gardner, Partha Pratim Talukdar, Jayant Krishnamurthy, and Tom Mitchell. Incorporating vector space similarity in random walk inference over knowledge bases. In *EMNLP*, 2014.

[53] Lise Getoor. Learning probabilistic relational models. In *SARA*. Springer, 2000.

[54] Lise Getoor and John Grant. Prl: A probabilistic relational language. *Machine Learning*, 2006.

[55] Lise Getoor and Ben Taskar. *Introduction to Statistical Relational Learning*. The MIT Press, 2007.

[56] Vibhav Gogate and Pedro M. Domingos. Exploiting logical structure in lifted probabilistic inference. In *StarAI Workshop at AAAI*, 2010.

[57] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *IR*, 4:133–151, 2001.

[58] Siddharth Gopal and Yiming Yang. Distributed training of large-scale logistic models. In *ICML*, 2013.

[59] Mourad Gridach, Hatem Haddad, and Hala Mulki. Churn identification in microblogs using convolutional neural networks with structured logical knowledge. In *ACL workshop on Noisy User-generated Text*, 2017.

[60] Martin Grohe, Kristian Kersting, Martin Mladenov, and Pascal Schweitzer. *An Introduction to Lifted Probabilistic Inference*, chapter Color Refinement and its Applications. Cambridge University Press, 2017.

[61] Benjamin Haeffele, Eric Young, and Rene Vidal. Structured low-rank matrix factorization: Optimality, algorithm, and applications to image processing. In *ICML*, 2014.

[62] Chih-Yang Hsia, Ya Zhu, and Chih-Jen Lin. A study on trust region update rules in newton methods for large-scale linear classification. In *ACML*, 2017.

[63] Tuyen N Huynh and Raymond J Mooney. Max-margin weight learning for markov logic networks. In *ECML*, 2009.

[64] Tuyen N. Huynh and Raymond J. Mooney. Online Max-margin Weight Learning with Markov Logic Networks. In *AAAI*, 2010.

[65] M. M. Islam, K. Mohammad Al Farabi, S. Sarkhel, and D. Venugopal. Scaling up inference in mlns with spark. In *Big Data*, 2018.

[66] Thorsten Joachims, Thomas Finley, and Chun-Nam John Yu. Cutting-plane training of structural svms. *Mach. Learn.*, 77:27–59, 2009.

[67] Henry Kautz, Bart Selman, and Yueyen Jiang. A general stochastic approach to solving problems with hard and soft constraints. In *SAT*, 1996.

[68] Seyed M Kazemi and David Poole. Knowledge compilation for lifted probabilistic inference: Compiling to a low-level language. In *KR*, 2016.

[69] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *NIPS*. 2017.

[70] Kristian Kersting. Lifted probabilistic inference. In *ECAI*, 2012.

[71] Kristian Kersting, Babak Ahmadi, and Sriraam Natarajan. Counting belief propagation. In *UAI*, 2009.

[72] Kristian Kersting, Youssef El Massaoudi, Babak Ahmadi, and Fabian Hadiji. Informed lifting for message-passing. In *AAAI*, 2010.

[73] Tushar Khot, Sriraam Natarajan, Kristian Kersting, and Jude Shavlik. Learning Markov logic networks via functional gradient boosting. In *ICDM*, 2011.

[74] Tushar Khot, Sriraam Natarajan, Kristian Kersting, and Jude Shavlik. Gradient-based boosting for statistical relational learning: the markov logic network and missing data cases. *Machine Learning*, 100(1):75–100, 2015.

[75] Dae Il Kim, Prem K Gopalan, David Blei, and Erik Sudderth. Efficient online inference for bayesian nonparametric relational models. In *NIPS*. 2013.

[76] Angelika Kimmig, Luc De Raedt, and Hannu Toivonen. Probabilistic explanation based learning. In *ECML*, 2007.

[77] Angelika Kimmig, Lilyana Mihalkova, and Lise Getoor. Lifted graphical models: a survey. *Machine Learning*, 99(1):1–45, 2015.

[78] Angelika Kimmig, Lilyana Mihalkova, and Lise Getoor. Lifted graphical models: a survey. *MLJ*, 99:1–45, 2015.

[79] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2014.

[80] Stanley Kok and Pedro Domingos. Learning Markov logic network structure via hypergraph lifting. In *ICML*, 2009.

[81] Stanley Kok and Pedro Domingos. Learning Markov logic networks using structural motifs. In *ICML*, 2010.

[82] Xiangnan Kong, Philip S. Yu, Ying Ding, and David J. Wild. Meta path-based collective classification in heterogeneous information networks. In *CIKM*, 2012.

[83] Arlind Kopliku, Karen Pinel-Sauvagnat, and Mohand Boughanem. Aggregated search: A new information retrieval paradigm. *ACM CS*, 46(3):41:1–41:31, 2014.

[84] Pigi Kouki, Shobeir Fakhraei, James Foulds, Magdalini Eirinaki, and Lise Getoor. HyPER: A flexible and extensible probabilistic framework for hybrid recommender systems. In *RecSys*, 2015.

[85] Pigi Kouki, Jay Pujara, Christopher Marcum, Laura M. Koehly, and Lise Getoor. Collective entity resolution in familial networks. In *ICDM*, 2017.

[86] Pigi Kouki, Jay Pujra, Christopher Marcum, Laura Koehly, and Lise Getoor. Collective entity resolution in multi-relational familial networks. *KAIS*, 2018.

[87] Pigi Kouki, James Schaffer, Jay Pujara, John O'Donovan, and Lise Getoor. Generating and understanding personalized explanations in hybrid recommender systems. 9, 2019.

[88] Pigi Kouki, James Schaffer, Jay Pujara, John O'Donovan, and Lise Getoor. Personalized explanations for hybrid recommender systems. In *IUI*, Marina del Ray, CA, USA, 2019. ACM.

[89] H. J. Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86(1):97–106, 1964.

[90] Sarasi Lalithsena, Sujan Perera, Pavan Kapanipathi, and Amit P. Sheth. Domain-specific hierarchical subgraph extraction: A recommendation use case. In *IEEE Big Data*, 2017.

[91] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *JMLR*, 18:1–52, 2018.

[92] Chih-Jen Lin and Jorge J. Moré. Newton's method for large bound-constrained optimization problems. *SIAM J. on Optimization*, 1999.

[93] Chih-Jen Lin, Ruby C Weng, and S Sathiya Keerthi. Trust region newton method for logistic regression. *JMLR*, 9(Apr):627–650, 2008.

[94] M. Lippi. Statistical relational learning for game theory. *T-CIAIG*, 8(4):412–425, 2016.

[95] Daniel Lizotte, Tao Wang, Michael Bowling, and Dale Schuurmans. Automatic gait optimization with gaussian process regression. In *IJCAI*, 2007.

[96] Daniel Lowd and Pedro Domingos. Efficient Weight Learning for Markov Logic Networks. In *KDD*, 2007.

[97] Sara Magliacane, Philip Stutz, Paul Groth, and Abraham Bernstein. foxPSL: A fast, optimized and extended psl implementation. *IJAR*, 67:111–121, 2015.

[98] George Marsaglia. Choosing a point from the surface of a sphere. *Ann. Math. Statist.*, 43(2):645–646, 1972.

[99] Ruben Martinez-Cantin, Nando de Freitas, Eric Brochu, José A. Castellanos, and Arnaud Doucet. A Bayesian exploration-exploitation approach for optimal online sensing and planning with a visually guided mobile robot. *Auton. Robots*, 27(2):93–103, 2009.

[100] Nickel Maxmilien, Murphy Kevin, Tresp Volker, and Gabrilovich Evgeniy. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2016.

[101] Rada F. Mihalcea and Dragomir R. Radev. *Graph-based Natural Language Processing and Information Retrieval*. Cambridge University Press, 2011.

[102] Lilyana Mihalkova and Raymond J Mooney. Bottom-up learning of Markov logic network structure. In *ICML*, 2007.

[103] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*. 2013.

[104] Martin Mladenov, Babak Ahmadi, and Kristian Kersting. Lifted linear programming. In *AIStats*, 2012.

[105] Martin Mladenov, Danny Heinrich, Leonard Kleinhans, Felix Gonsior, and Kristian Kersting. Reloop: A python-embedded declarative language for relational optimization. In *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[106] Martin Mladenov and Kristian Kersting. Equitable partitions of concave free energies martin. In *UAI*, 2015.

[107] Martin Mladenov, Kristian Kersting, and Amir Globerson. Efficient lifting of MAP LP relaxations using k-locality. In *AIStats*, 2014.

[108] Martin Mladenov, Leonard Kleinhans, and Kristian Kersting. Lifted inference for convex quadratic programs. In *AAAI*, 2017.

[109] Jonas Mockus. On Bayesian Methods for Seeking the Extremum and their Application. In *IFIP Congress*, 1977.

[110] Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. The application of Bayesian methods for seeking the extremum. In *TGO*, 1978.

[111] Mervin E. Muller. A note on a method for generating points uniformly on n-dimensional spheres. *Commun. ACM*, 2(4):19–20, 1959.

[112] Houssam Nassif, Yirong Wu, David Page, and Elizabeth S. Burnside. Logical differential prediction bayes net, improving breast cancer diagnosis for older women. In *AMIA*, 2012.

[113] Sriraam Natarajan, Prasad Tadepalli, Eric Altendorf, Thomas G Dietterich, Alan Fern, and Angelo Restificar. Learning first-order probabilistic models with combining rules. In *ICML*, 2005.

[114] Aniruddh Nath and Pedro Domingos. Efficient lifting for online probabilistic inference. In *AAAI Workshop on StarAI*, 2010.

[115] Jennifer Neville and David Jensen. Relational dependency networks. pages 653–692, 2007.

[116] Feng Niu, Christopher Ré, AnHai Doan, and Jude Shavlik. Tuffy: Scaling up statistical inference in Markov logic networks using an rdbms. In *VLDB*, 2011.

[117] George Papadakis, Jonathan Svirsky, Avigdor Gal, and Themis Palpanas. Comparative analysis of approximate blocking techniques for entity resolution. *PVLDB Endow.*, 9(9):684–695, 2016.

[118] Singla Parag and Domingos Pedros. Entity resolution with markov logic. In *ICDM*, 2006.

[119] David Poole. First-order probabilistic inference. In *IJCAI*, 2003.

[120] Hoifung Poon and Pedro Domingos. Sound and efficient inference with probabilistic and deterministic dependencies. In *AAAI*, 2006.

[121] Jay Pujara, Eriq Augustine, and Lise Getoor. Sparsity and noise: Where knowledge graph embeddings fall short. In *EMNLP*, 2017.

[122] Jay Pujara, Ben London, and Lise Getoor. Budgeted online collective inference. In *UAI*, 2015.

[123] Jay Pujara, Ben London, Lise Getoor, and William Cohen. Online inference for knowledge graph construction. In *StarAI*, 2015.

[124] Jay Pujara, Hui Miao, Lise Getoor, and William Cohen. Knowledge graph identification. In *ISWC*, 2013.

[125] Motakuri V. Ramana, Edward R. Scheinerman, and Daniel Ullman. Fractional isomorphism of graphs. *Discrete Mathematics*, 1994.

[126] Nandini Ramanan, Gautam Kunapuli, Tushar Khot, Bahare Fatemi, Seyed Mehran Kazemi, David Poole, Kristian Kersting, and Sriraam Natarajan. Structure learning for relational logistic regression: An ensemble approach. In *KRR*, 2018.

[127] Arti Ramesh, Mario Rodriguez, and Lise Getoor. Multi-relational influence models for online professional networks. In *WI*, 2017.

[128] Nikhil Rao, Hsiang-Fu Yu, Pradeep K Ravikumar, and Inderjit S Dhillon. Collaborative filtering with graph information: Consistency and scalable methods. In *NIPS*, 2015.

[129] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[130] Matthew Richardson and Pedro Domingos. Markov logic networks. *MLJ*, 62:107–136, 2006.

[131] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[132] Somdeb Sarkhel, Parag Singla, and Vibhav Gogate. Fast lifted map inference via partitioning. In *NIPS*, 2015.

[133] Somdeb Sarkhel, Deepak Venugopal, Tuan Anh Pham, Parag Singla, and Vibhav Gogate. Scalable training of Markov logic networks using approximate counting. In *AAAI*, 2016.

[134] Somdeb Sarkhel, Deepak Venugopal, Parag Singla, and Vibhav Gogate. Lifted MAP inference for Markov logic networks. In *AIStats*, 2014.

[135] Prithviraj Sen, Amol Deshpande, and Lise Getoor. Bisimulation-based approximate lifted inference. In *UAI*, 2009.

[136] Prithviraj Sen, Galileo Mark Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. volume 29, pages 93–106, 2008.

[137] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[138] Jude Shavlik and Sriraam Natarajan. Speeding up inference in Markov logic networks by preprocessing to reduce the size of the resulting grounded network. In *IJCAI*, 2009.

[139] Parag Singla and Pedro Domingos. Discriminative Training of Markov Logic Networks. In *AAAI*, 2005.

[140] Parag Singla and Pedro M. Domingos. Lifted first-order belief propagation. In *AAAI*, 2008.

[141] Parag Singla, Aniruddh Nath, and Pedro Domingos. Approximate lifting techniques for belief propagation. In *AAAI*, 2014.

[142] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In *NIPS*, 2012.

[143] Dhanya Sridhar, Shobeir Fakhraei, and Lise Getoor. A probabilistic approach for collective similarity-based drug-drug interaction prediction. *Bioinformatics*, 32(20):3175–3182, 2016.

[144] Dhanya Sridhar, Jay Pujara, and Lise Getoor. Scalable probabilistic causal structure discovery. In *IJCAI*, 2018.

[145] Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *ICML*, 2010.

[146] Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias W. Seeger. Information-theoretic regret bounds for gaussian process optimization in the bandit setting. information theory. *IEEE Transactions on*, page 3265, 2012.

[147] Sriram Srinivasan*, Eriq Augustine*, and Lise Getoor. Tandem inference: An out-of-core streaming algorithm for very large-scale relational inference. In *AAAI*, 2020.

[148] Sriram Srinivasan, Behrouz Babaki, Golnoosh Farnadi, and Lise Getoor. Lifted hinge-loss Markov random fields. In *AAAI*, 2019.

[149] Sriram Srinivasan, Golnoosh Farnadi, and Lise Getoor. Bayesian optimization for weight learning in probabilistic soft logic. In *AAAI*, 2020.

[150] Sriram Srinivasan, Nikhil S Rao, Karthik Subbaian, and Lise Getoor. Identifying facet mismatches in search via micrographs. In *CIKM*, 2019.

[151] Charles Sutton and Andrew McCallum. An introduction to conditional random fields. *FTML*, 4:267–373, 2012.

[152] Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin markov networks. In *NIPS*, 2003.

[153] W.R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3–4):285—294, 1933.

[154] Ingo Thon, Niels Landwehr, and Luc De Raedt. Stochastic relational processes: Efficient inference and applications. *MLJ*, 82(2):239–272, 2011.

[155] Deepak Venugopal and Vibhav Gogate. Evidence-based clustering for scalable inference in markov logic. In *ECML*, 2014.

[156] Deepak Venugopal and Vibhav Gogate. Evidence-based clustering for scalable inference in Markov logic. In *ECML*, 2014.

[157] Deepak Venugopal, Somdeb Sarkhel, and Vibhav Gogate. Just count the satisfied groundings: scalable local-search and sampling based inference in MLNs. In *AAAI*, 2015.

[158] Deepak Venugopal, Somdeb Sarkhel, and Vibhav Gogate. Magician: Scalable inference and learning in markov logic using approximate symmetries. Technical report, Department of Computer Science, The University of Memphis, 2016.

[159] Q. Wang, Z. Mao, B. Wang, and L. Guo. Knowledge graph embedding: A survey of approaches and applications. *TKDE*, 29(12):2724–2743, 2017.

[160] Wei-Chung Wang and Lun-Wei Ku. Identifying chinese lexical inference using probabilistic soft logic. In *ASONAM*, 2016.

[161] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Peng Cui, P. Yu, and Yanfang Ye. Heterogeneous graph attention network. In *ICWC*, 2019.

[162] Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando De Freitas. Bayesian optimization in a billion dimensions via random embeddings. *J. Artif. Int. Res.*, 55(1):361–387, 2016.

[163] Zhuoyu Wei, Jun Zhao, Kang Liu, Zhenyu Qi, Zhengya Sun, and Guanhua Tian. Large-scale knowledge base completion: Inferring via grounding network sampling over selected instances. In *CIKM*, 2015.

[164] Benyu Zhang, Hua Li, Yi Liu, Lei Ji, Wensi Xi, Weiguo Fan, Zheng Chen, and Wei-Ying Ma. Improving web search results using affinity graph. In *SIGIR*, 2005.

[165] Yizhou Zhang, Yun Xiong, Xiangnan Kong, Shanshan Li, Jinhong Mi, and Yangyong Zhu. Deep collective classification in heterogeneous information networks. In *WWW*, 2018.

[166] Xiaojin Zhu, Andrew Goldberg, Jurgen Van Gael, and David Andrzejewski. Improving diversity in ranking using absorbing random walks. In *ACL*, 2007.