Learning with Degree-Based Subgraph Estimation

Bert Huang

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2011

ABSTRACT

Learning with Degree-Based Subgraph Estimation

Bert Huang

Networks and their topologies are critical to nearly every aspect of modern life, with social networks governing human interactions and computer networks governing global information-flow. Network behavior is inherently structural, and thus modeling data from networks benefits from explicitly modeling structure. This thesis covers methods for and analysis of machine learning from network data while explicitly modeling one important measure of structure: degree.

Central to this work is a procedure for exact maximum likelihood estimation of a distribution over graph structure, where the distribution factorizes into edge-likelihoods for each pair of nodes and degree-likelihoods for each node. This thesis provides a novel method for exact estimation of the *maximum likelihood* edge structure under the distribution. The algorithm solves the optimization by constructing an augmented graph containing, in addition to the original nodes, auxiliary nodes whose edges encode the degree potentials. The exact solution is then recoverable by finding the maximum weight b-matching on the augmented graph, a well-studied combinatorial optimization.

To solve the combinatorial optimization, this thesis focuses in particular on a belief propagation-based approach to finding the optimal b-matching and provides a novel proof of convergence for belief propagation on the loopy graphical model representing the b-matching objective. Additionally, this thesis describes new algorithmic techniques to improve the scalability of the b-matching solver.

In addition to various applications of node degree in machine learning, including classification and collaborative filtering, this thesis proposes a learning algorithm for learning the parameters of the distribution from network data consisting of node attributes and network connectivity, using strategies similar to maximum-margin structured prediction.

The main methods and results in this thesis represent a deep exploration of exact degree-based estimation for machine learning from network data, and furthermore lead to various extensions and applications of the main idea described within.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgments

I owe my deepest gratitude to various people who made this thesis possible. Because of their direct influence on my research, I am grateful to my advisors, Tony Jebara, who advised me on all of the work in this thesis, and Ansaf Salleb-Aouissi, who advised me on a significant amount of research that does not fit in the compact topic area covered in this thesis. I am proud to now be part of each of their academic lineages. I am also grateful to David Waltz and the Center for Computational Learning Systems for providing me with the opportunity to enter the Ph.D. program by hiring me as a graduate research assistant my first few years in the program. Other people who deserve thanks include the members of the Machine Learning Laboratory, which over the years that I was a member included Stuart Andrews, Delbert Dueck, Andrew Howard, Risi Kondor, Darrin Lewis, Raphael Pelossof, Blake Shaw, Pannaga Shivaswamy, Yingbo Song, Kapil Thanadi, and Adrian Weller.

Before my time as a Ph.D. student, I was fortunate to have brilliant, enthusiastic teachers at all levels of my education. At Brandeis University, as an undergraduate, my love for artificial intelligence and machine learning was especially nurtured by Jordan Pollack, Edwin de Jong, and Jerry Samet. As a graduate student at Columbia, my passion for my research area was further honed by Rocco Servedio, John Kender and, of course, Tony. The work these professors did has had a profound impact on the way I think, and they deserve sincere thanks for that effort.

I additionally thank Augustin Chaintreau, Devavrat Shah, and Cliff Stein for serving on my thesis committee, Lexing Xie for mentoring me during my internship at IBM Research, and Michele Merler, who shares an office with me and Blake Shaw and therefore had to put up with all our hijinks.

As a result of the work in this thesis, I owe a tremendous debt of friendship to all of my friends who are not named “Bert’s Thesis”, which, in fact, is all of them. You have not received the attention from me that you deserve, and yet I continually feel your support.

Most importantly, my family, to whom this thesis is dedicated, deserves the most thanks. Dur-

ing the final few months of my Ph.D. program, my grandmother, Chen Chang, passed away after years of suffering from dementia. Since she lived with my family and played a significant role in raising me, she was profoundly influential on me as a person, and her neurological illness which manifested when I was becoming an adult was influential in my choosing to study the science of information. My parents, Dr. Shaw Huang[1] and Phoebe Phong Chang, provided me with the values that guided me toward science, writing, and hard work. My sister and my brother-in-law, Angela and Al Ming, were among other things extremely influential in my studying computer science. My brother, Christopher Huang, has vision, artistry, and skills as a photographer[2] that inspire me every day to seek the beauty in my own work. My future family-in-law, Michael, Roberta, and Danielle Smith, took me in as a member of their family even though I (a) took forever to propose, and (b) have undying love for the Red Sox. And, finally, immeasurable gratitude goes to the woman with whom I share everything except for *whatever-the-heck-all-the-math-in-this-thesis-means*, who is the completion of the *strange loop* that is my identity, my fiancée Emily Smith.

Bert Huang, July 2011

[1] While many other people I thank here have doctorates, I list my father as "Dr. Huang" because upon publication of this thesis by Columbia University, I will ask people to "please call me Bert; Dr. Huang is my dad."

[2] `http://www.christopherhuang.com`

For my family

Chapter 1

Introduction

Networks are collections of objects, or *nodes*, that interact with each other. Networks govern nearly every aspect of the modern world. Computer networks form the backbone of civilization, controlling the flow of information. With the advent of Internet-based social networking services in the early 21st century, the commercial and industrial interest in analysis of social networks has burgeoned [Boyd and Ellison, 2007]. The most critical feature of any network is its *structure* or *topology*, which, while can be described as a list of connections between nodes, cannot be fully characterized without considering many, if not all, connections at once. For this reason, efficient computational models that incorporate structure are uncommon. Many notions of structure are too complex to handle efficiently and exactly. One relatively simple measure of structure is the local *degree* of each node: the number of connections or interactions of each node. This thesis is an exploration of methods exploiting the node-degree aspect of structure for *machine learning* in network settings.

Machine learning is the task of extracting information from raw data. The extracted information often consists of a model from which predictions or decisions can be made. Combining ideas from the fields of statistics and artificial intelligence, machine learning techniques often rely on known independence between parts of data. For example, in the traditional setting of *supervised learning*, data points are often assumed to be independent. When data comes from real networks, the structural nature of networks makes entity independence unlikely, and the assumption of too much independence in a mathematical model often leads to severe deficiencies. In contrast, the assumption of too little independence results in overly complex models for which computational learning is either expensive or intractable. It is thus important to identify models of structure that

are both efficiently learnable and yet structural enough to capture the behavior of real networks.

Since machine learning techniques use statistical intuitions and mathematical measures of uncertainty, they often have associated *probabilistic models* (either explicitly or implicitly). Probabilistic models are powerful formalisms for data analysis in part because they make modeling assumptions explicit. This thesis revolves around a probabilistic model for networks that explicitly depends on the degrees of nodes.

The degree of a node in a network is a local measure of structure, in the sense that each node "knows" its own degree just by knowing its neighbors. Other measures of structure, such as *triangles*, *path lengths*, or *cliques* require exploration beyond local information to compute. Despite the local measure of degree, the value of the degree depends on aggregating information about all the interactions and non-interactions between nodes. Furthermore, the degree distribution of a graph is a well-studied global measure of structure that has played a central role in previously established work in network analysis [Barabási, 2003].

1.1 Thesis contributions

The contributions in this thesis include a novel technique for predicting the most likely network structure given a probabilistic model factorized into edge likelihoods and degree likelihoods. This estimation technique, viewed outside the probabilistic interpretation, is a generalization of *maximum weight b-matching*, which is an efficiently solvable combinatorial problem, itself a generalization of the more commonly studied *maximum weight matching*.

The optimization procedure for the graph structure estimation above is driven by reduction to a maximum weight b-matching problem, for which this thesis provides a new solver based on *loopy belief propagation*. Loopy belief propagation is a heuristic approximation algorithm for probabilistic inference, which is not guaranteed to find a true optimum or even converge on arbitrary problems. However, this thesis provides a proof of efficient convergence to the optimum of the maximum weight b-matching problem, which defines one of few known classes of problems on which loopy belief propagation is guaranteed to converge to the true solution. This thesis also provides scalability improvements for belief propagation that make it a competitively fast solver for certain b-matching problems.

This thesis contains explorations of application areas using the procedure above and its extensions. Among these applications is the task of *collaborative filtering*, for which two new approaches are derived and tested. In collaborative filtering, the goal is to predict user preferences for items from past data. Each of these methods performs the prediction task by leveraging a notion of degree, counting the types of recommendations given by users and received by items.

Parameterizing the components of the edge-degree probabilistic model, this thesis also provides a general learning algorithm that learns parameters from data, allowing prediction of graph structure for new node data when edge and degree likelihoods are not given *a priori*. The learning algorithm is a new instance of *structured prediction*, and a stochastic variant is provided for learning from large-scale data sets.

Example experiments, discussion and analysis of the techniques and models are provided throughout the text.

1.2 Organization

Terminology and notation This thesis uses the term *network* to describe a real set of interacting objects, and the term *graph* to describe the mathematical formalism often used to model networks. Often the distinction is unnecessary, in which case the terms are used interchangeably. Unless explicitly stated, all uses of the verb "learn" and its conjugations refer to machine or algorithmic learning, in the sense of a computational technique for extracting information from data.

Each chapter of this thesis is meant to be as self-contained as possible, with reintroduction of mathematical notation when appropriate. Symbol names are somewhat consistent, though conventions such as using the symbol x in various typefaces as observed data, n and m for the cardinalities of sets, $\mathbf{U}$ and $\mathbf{V}$ as matrix factors, the symbols G, V, U, and E for describing graphs in set notation, and index symbols mostly take precedence over consistency across chapters, especially when chapters address different problem settings. Notation within each chapter, however, is always consistent. Bold uppercase symbols such as $\mathbf{X}$ denote matrices, and bold lowercase symbols such as $\mathbf{x}$ denote vectors. The indexed lowercase $\mathbf{x}_i$ is used to denote the i'th row (or column) of matrix $\mathbf{X}$, in uppercase. Ranges for index terms (usually i, j, or k) are often omitted for cleanliness when the range is obvious (1 to n is a good guess).

The *graph* formalism A *graph* is a mathematical formalism which naturally represents a network. A graph consists of a set of nodes and a set of *edges*, or connections, between nodes. Graphs are commonly notated using either *set notation*, where a graph is represented by a pair $G = \{V, E\}$ composed of node-set V and edge-set E, or alternatively *matrix notation*, in which a graph topology is represented by a binary *adjacency matrix* $\mathbf{A}$ whose entries A_{ij} are 1 if the edge between the i'th and the j'th nodes is present and zero if no edge is present.

Set notation is preferred when the unordered nature of the nodes is emphasized, and furthermore set notation emphasizes the sparse nature of graphs. In the set notation, the quantity of named information is determined by the number of nodes and edges in the graph, whereas the matrix notation represents the presence or absence of an edge for each pair of nodes. Regardless of the notational choice, conversion between representations is relatively straightforward in theory and practice[3], and this thesis uses both notational forms in its different chapters.

Outline The remainder of the thesis is segmented into two parts. Part I describes the main degree-based probabilistic model and the estimation procedure for the model, and Part II describes applications of the model to different machine learning problems. In Part I, Chapter 2 summarizes the current state of knowledge on the problem of *link prediction*, which is the primary problem the methods in this thesis address. Chapter 3 introduces the degree-based probabilistic model and details the reduction algorithm that converts estimation of the most likely graph structure to a maximum weight b-matching. Chapter 4 describes a belief propagation solver for maximum weight b-matchings, including a convergence proof and scalability improvements. In Part II, Chapter 5 describes applications of the techniques from Part I to more traditional machine learning problems of supervised and semi-supervised classification. Chapter 6 describes the application of degree-based learning to the problem of collaborative filtering using degree priors derived from concentration of data statistics. Chapter 7 describes a learning algorithm to fit model parameters for the degree-based graph distribution from training data and applies the learned model to predicting real networks. Finally, Chapter 8 concludes with a summary of the work described in the thesis and discussion of open questions and future work.

[3]Conversion between representations is especially straightforward in computer implementations with the availability of sparse matrix libraries.

The appendices include extensions, proofs, and derivations. Appendix A contains a study of a novel approximation algorithm for the matrix permanent derived from the belief propagation solver for matchings and some additional heuristics for efficient implementation of the belief propagation algorithm from Chapter 4. Appendix B contains full details for the belief propagation convergence proof as well as the deviation bound proofs for collaborative filtering degree priors. Appendix C provides the derivation for the dual formulation of the collaborative filtering algorithm proposed in Chapter 6.

Part I

Degree-Based Subgraph Estimation

Chapter 2

Link Prediction

The class of problems known as *link prediction* includes tasks surrounding the estimation of network topology when true topology is uncertain. The topology may be uncertain because it is unobserved or because observed interactions are not trusted as ground-truth. This section describes various definitions and techniques for link prediction and their implications with respect to the contributions of this thesis.

2.1 Problem formulations for link prediction

The problem of link prediction has been studied in various forms, some of which are summarized in this section. The problem formulation often determines what assumptions can be made about the data and thus what models and learning algorithms are appropriate. In general, since connections in networks represent relationships between node entities, machine learning from network data is often referred to as *relational learning* [Getoor and Taskar, 2007]. Using relational learning, various link prediction problem settings are described below.

Network completion In some applications, the network is partially observed, and the prediction algorithm attempts to estimate the rest of the network, or perform *network completion*. For example, *link recommendation* is the task of suggesting connections to users in a social network or authors of documents in a citation network. Often the observed part of the network is a "snapshot" of a network at a point in time, and the prediction task is to estimate the network topology in the future.

In other applications, the prediction algorithm is tasked to predict the structure of an entire network. In these problems, auxiliary information is typically available, such as the attributes of the nodes involved. A variant of network prediction is a problem in which the observed connectivity in data is not trusted, and instead considered noisy edge measurements, in which case the prediction task is to estimate the true underlying structure.

Between these two related problems is the task of predicting a partially-observed network when new, previously unseen nodes join the network. While the nodes in the network that already have some partially-observed connections are mostly analogous to nodes in a pure network completion problem, the new nodes are more analogous to the nodes in a pure network prediction problem. Unfortunately, practical application of link prediction problems tends to fall in this hybrid setting.

In each of these problems, a useful initial step is to understand and formalize what observed edges mean, or, more importantly, what unobserved edges mean. In practice, observed edges occasionally do not correctly indicate connectivity. Often a distinction is desired between measured interactions such as users sharing links on a social networking service and some underlying connectivity such as "friendship". More importantly, the absence of an interaction observation rarely indicates the absence of a connection, particularly because measured data is typically measured over a period of time, during which connected nodes simply may not interact.

Collaborative filtering and multi-relational data The problem of *collaborative filtering* is to predict user preferences for items given past interaction data. Typically, the data available is a measurement of user tastes for items, such as ratings or reviews on an online shopping service. Collaborative filtering techniques are commonly applied to implement *recommendation engines*, which suggest items to users based on their previous history. Machine learning techniques for collaborative filtering segment the problem into two settings: *weak* and *strong generalization* [Marlin, 2004]. For weak generalization, predictions are made for users and items already seen in data, whereas strong generalization involves predictions for new users and new items from which measurements are not yet available.

As in network prediction and completion, problem settings are further differentiated by what type of data is available. In particular, many approaches assume only interaction data is available, while some assume that additional data is available about the users and items.

The collaborative filtering problem is a form of network prediction, where the bipartite network between user and item nodes is partially observed and completed by an algorithm. The edges between users and items can be either weighted or typed, depending on the interpretation. Thus, collaborative filtering can be viewed as estimating the multi-relational structure between users and items when multiple relations, *e.g.*, rating scores or tags, are possible.

2.2 Prediction methods

Graph prediction and its variants and extensions have been studied extensively by various fields, including computer science, statistics, sociology, biology and applied mathematics. This section summarizes some important methods from established literature, but coverage of methods is biased toward relationships to the methods and ideas of this thesis, and thus this section is not a comprehensive review of all methods.

2.2.1 Graph distance methods

Given a partially-observed graph, measures of graph distance are available as estimates of node similarity. For example, the length of the shortest path between any two nodes is a natural measure. While the shortest path is quite brittle in the sense that it can change drastically with small changes in the network, measures of connected distance based on randomness are considerably more robust. For example, the *hitting time* is the measure of the expected time for a random walk from one node to another, and the *commute time* is the random-walk time from a node to another and back. A random walk is usually defined as a process that maintains a pointer to a current node and advances to the next step by randomly choosing a next node uniformly from the current node's neighbors. The *Katz measure* is a weighted sum of all paths between nodes, where paths are weighted according to their length, valuing shorter paths exponentially more [Liben-Nowell and Kleinberg, 2007].

Relatedly, measures on the neighborhoods of nodes themselves are also natural similarity functions. For example, the *Jaccard coefficient*, which is a normalized count of common features, can be computed between nodes by using neighbor nodes as features [Liben-Nowell and Kleinberg, 2007].

2.2.2 Structured prediction approaches

The general problem of graph estimation, estimating graph structure from data, is a form of *structured prediction*. Structured prediction problems have dependencies between output variables, in contrast to independent labels, whether or not the input data is independent. More specific examples of such problems include matchings for image feature alignment [Belongie *et al.*, 2002; Caetano *et al.*, 2009], word alignments in machine translation [Matusov *et al.*, 2004], natural language parsing [Koo *et al.*, 2007] and predicting graph structure. The distinguishing challenge presented by structured output is that the output state space has many dependencies that couple variables together such that they cannot be considered independently. Structured outputs are becoming increasingly important as independence assumptions are becoming less realistic in modern data problems. One example of an effective approach for structured prediction is the *structural support-vector machine*, which solves a quadratically regularized objective subject to exponentially many constraints defined by the structured output. Approaches for handling the exponentially many constraints typically involve using a *separation oracle*, which is an efficient procedure to find the worst-violated constraint given a current parameter estimate. The worst-violated constraint can then be added to a working set that bounds the feasible region [Joachims *et al.*, 2009], or may be used to define a subgradient step in a stochastic optimization [Ratliff *et al.*, 2007].

2.2.3 Latent space models

A large class of network-modeling approaches aims to map nodes of a network to points in *latent space*. These models often use the assumption that the probability of an edge-interaction is conditionally independent given the endpoint nodes, but dependencies arise because the endpoint nodes are unknown latent variables.

Many latent space models are fully parameterized probabilistic models [Airoldi *et al.*, 2008; Xing *et al.*, 2010], though non-parametric extensions have been studied [Miller *et al.*, 2009; Ho *et al.*, 2011]. The *mixed-membership stochastic blockmodel* (MMSB) [Airoldi *et al.*, 2008] models each node as a multinomial distribution over clusters, and edge connectivity is determined by the cluster membership of node pairs. Exact prediction in these models is computationally intractable, but approximate inference is possible using sampling methods such as Gibbs and *Markov chain Monte-Carlo* sampling or variational approximation such as *structured* and unstructured *mean-field*.

The MMSB model shares many similarities with the *latent Dirichlet allocation* (LDA) topic-model [Blei *et al.*, 2003], which models text documents as distributions over latent topics. The LDA model itself has various extensions that incorporate network structure such as the *relational topic model* [Chang and Blei, 2010] and *networks uncovered by Bayesian inference* (NUBBI) [Chang *et al.*, 2009], which extend LDA by including parameterized edge-likelihood factors in the LDA probability distribution.

Additionally, matrix factorization techniques popularly used for collaborative filtering [Bell and Koren, 2007; Koren *et al.*, 2009; Rennie and Srebro, 2005; Salakhutdinov and Mnih, 2008a; Salakhutdinov and Mnih, 2008b; Srebro *et al.*, 2004] are examples of latent space estimation. Given a *rating matrix* of scores input by user rows for item columns, these techniques estimate factors for the rating matrix such that multiplying the factors maps to observed rating data, and by imposing regularization or prior probabilities on the factor matrices, generalize to unseen entries in the rating data.

2.2.4 Exponential random graph models

Exponential random graph models (ERG) are prominent in social network analysis [Robins *et al.*, 1999; Robins *et al.*, 2006; Robins *et al.*, 2007]. Also known as $p*$ models, these generative models represent the probability of a graph structure as a factorized probability over weighted *graph motif* statistics. A graph motif is a countable structural pattern that may occur in the network, such as reciprocated directed links, triangles, stars or cliques of different cardinalities. Since these statistics are structural, various dependencies between edges arise, and computation of the likelihood is computationally intractable. Thus, various approximation approaches are available for fitting ERG models. Of note is the maximum *pseudo-likelihood* approach, in which the pseudo-likelihood is used as an approximation of the likelihood and maximized to select model parameters. Pseudo-likelihood is computed by considering only the conditional likelihood of each edge variable conditioned on all other variables taking their observed state. Other strategies for fitting ERG model parameters include sampling-based approaches.

2.2.5 Other methods

Various generative models for graphs have been suggested in the literature of overlapping fields of research. Some notable models include the Erdos-Renyi model [Erdos and Renyi, 1959], in which edge presence is identically and independently distributed (*iid*), the Strogatz-Watts model [Newman *et al.*, 2001], in which nodes are initially connected on a lattice and edges are randomly swapped, and the *preferential attachment* model [Albert and Barabási, 2002], in which nodes iteratively join the network, and the likelihood of attachment to existing nodes is proportional to the existing nodes' degrees.

The recently proposed *Kronecker graph model* is an approach for modeling graphs based on matrix operations. The Kronecker generative graph model describes a process of building a graph using repeated Kronecker matrix multiplication, the results of which are proven to obey many known graph properties [Leskovec *et al.*, 2010].

Since edge measurements can themselves be noisy, structural modeling of networks can be used to denoise these measurements. For example, approximate inference with degree priors has been used to smooth noisy edge observations in biological networks [Morris and Frey, 2003].

2.3 Link prediction summary

Most of the network modeling approaches summarized in this section handle structure by resorting to approximations. The degree-based estimation technique studied in this thesis solves its structure-sensitive objective exactly. The estimation procedure is also flexible in that its edge-likelihoods may be parameterized or derived from many of the established graph models described in this section, and thus is complementary to existing approaches.

The graph distance ideas can be used directly to determine edge-likelihoods and combined with degree likelihoods in the proposed model, as can the likelihoods estimated from latent space models. The probabilistic model used in this thesis is related to the exponential random graph model, but inherent differences include the modeling of individual nodes, rather than the anonymous nodes of ERG models, and the structural motifs being limited to degree counts.

Chapter 3

Degree-Based Subgraph Estimation

Given an input graph, a common task is to prune the input graph to form a smaller graph with desirable properties. This chapter describes a procedure for pruning a graph that optimizes an objective dependent on the edges and the node degrees in the pruned graph. The problem can be interpreted as a generalization of the maximum weight b-matching in combinatorial optimization, or as maximum likelihood inference in a probabilistic model of graph structure.

From a pure optimization viewpoint, an efficient procedure for obtaining an optimal subgraph is useful for classical applications such as *resource allocation*, in which restrictions or costs may be associated with the degrees of nodes in addition to rewards of edge connectivity. In resource allocation problems, nodes correspond to both resource producers and consumers, and edges may correspond to assignments between the producers and consumers. Physical or economic restrictions may limit or penalize producers who are assigned too many or too few consumers, and vice versa, leading to a natural application of degree-based subgraph estimation. Well-studied formulations for these problems, such as the *linear assignment problem*, allow hard-constraints on degrees, and the methods in this chapter generalize these formulations to allow penalized degrees, or soft-constraints.

The objective function optimized in this chapter is linear with respect to the binary representation of the edges and indicator functions for the degrees of each node. This property allows a natural interpretation of the optimization as a *maximum likelihood* estimation of a log-linear model.

The main problem is referred to as a *subgraph estimation*, as opposed to *graph estimation* to emphasize the possibility of sparsity in the input graph. If edges are known to be impossible, they may be omitted in the input graph, alleviating computational load of the solver algorithm. If,

however, graph estimation is desired, in which all edges are possible, a fully-connected graph may be the input, allowing the estimation procedure to choose from all possible edges.

The remainder of the chapter first introduces the objective, then details the procedure for optimizing the objective. The solver reduces the problem to an augmented maximum weight b-matching, which various known algorithms solve.

3.1 Maximum weight matching and its generalizations

This section introduces the classical problem of *maximum weight matching*, its generalizations and variants, leading to the generalization of interest, which will be referred to as *degree-based matching*. First, we define a general framework for matching-like problems. Given a weighted, undirected graph $G = \{V, E, w\}$, where E is a set of node pairs $(u, v) \in E$, $u, v \in V$, and $w(e) \mapsto \mathbb{R}$ is a function that maps edges to their weights, a matching-like problem is solved by finding the *induced subgraph* of G that has maximum weight subject to requirements dependent on the degrees of nodes in $\hat{G}$. An induced subgraph $\hat{G} = \{V, \hat{E}, w\}$ is a graph that shares the same node set V with G but whose edge-set $\hat{E} \subseteq E$ is a subset of the original edge-set E. Thus, each problem is an optimization of the form

$$\hat{E} = \underset{E' \subseteq E}{\operatorname{argmax}} \sum_{(u,v) \in E'} w(u, v) \text{ s.t. } C(E'),$$

where C is a set of requirements dependent on the degrees. In the standard *maximum-weight matching* problem, the degree requirements are that each degree is no more than one, or in the *maximum-weight perfect matching* problem, the requirements are that each degree is exactly one.

Generalized matching extends maximum weight matching to allow degrees other than one. In *b-matching* problems, the maximum and target degrees are defined by a set of non-negative integers b_v for each $v \in V$, such that in the *maximum-weight b-matching problem*, each node v must have degree no more than b_v and in the *maximum-weight perfect b-matching* problem, each node v must have exactly degree b_v. Figure 3.1 illustrates a bipartite b-matching, in which the nodes are divided into two bipartitions and edges only exist between bipartitions.

A further generalization of b-matching is *degree constrained subgraph* (DCS), in which each node's degree is subject to a lower and upper bound. Thus, degree constrained subgraphs subsume

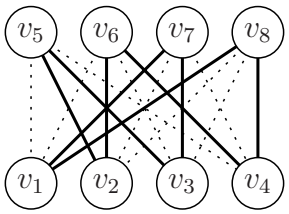


Figure 3.1: Example bipartite b-matching. Dashed lines represent possible edges, solid lines represent b-matched edges. In this case $b_v = 2$ for all nodes.

perfect and non-perfect b-matchings, *e.g.*, the perfect b-matching problem can be represented by setting the lower and upper bounds to be equal. DCS also generalize *b-covers*, which are graphs where nodes have at least b adjacent edges. Table 3.1 lists the constraints for each of these generalized matching problems.

Problem	Standard constraints		Perfect constraints	
matching	$\deg(v, E') \leq 1,$	$\forall v \in V$	$\deg(v, E') = 1,$	$\forall v \in V$
b-matching	$\deg(v, E') \leq b_v,$	$\forall v \in V$	$\deg(v, E') = b_v,$	$\forall v \in V$
b-cover	$\deg(v, E') \geq b_v,$	$\forall v \in V$	$\deg(v, E') = b_v,$	$\forall v \in V$
DCS	$b_v \leq \deg(v, E') \leq b'_v,$	$\forall v \in V$		

Table 3.1: Constraints for maximum-weight generalized matching problems. Each problem is solved by maximizing the total weights of active edges subject to the constraints listed above. The function $\deg(v, E')$ is the degree of node v in edge-set E', *i.e.*, the number of edges adjacent to v.

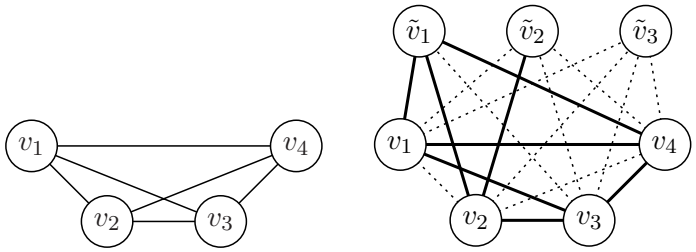
3.1.1 Degree-based matching

A larger class of problems is the *maximum-weight degree-based induced subgraph* problem, or *degree-based matching*, where the set of feasible induced subgraphs is unconstrained, but instead penalized according to functions of node degrees. The problem class is so named because these functions are interpretable as *degree-based* potentials in a probabilistic interpretation of the maxi-

mum weight induced subgraph problem. Degree-based matching problems are of the form,

$$\hat{E} = \underset{E' \subseteq E}{\operatorname{argmax}} \sum_{(u,v) \in E'} w(u,v) + \sum_{v \in V} \psi_v(\deg(v, E')). \tag{3.1}$$

The function $\deg(v, E)$ indicates the number of neighbors for node v in edge set E'. This class of problems contains all feasible DCS problems, since any DCS problem can be solved by defining appropriate ψ degree functions that return zero for all feasible degrees and negative infinity for all infeasible degrees. However, even though DCS is subsumed by degree-based matching, for concave degree functions ψ, degree-based matching is no harder than DCS. The remainder of this section describes a procedure that solves degree-based matching by reducing the problem to an augmented DCS problem.



(a) Original graph (b) Augmented graph

Figure 3.2: Example of constructing the augmented graph using auxiliary nodes to encode degree preferences. On the left, we are to find a subgraph of a fully connected, four-node graph. On the right, the augmented graph contains auxiliary nodes. An example b-matching for the original nodes, which includes edges between original nodes and auxiliary nodes, is marked with bold lines.

Reduction of degree-based matching to maximum weight b-matching In instances where the degree preference functions ψ are concave, a procedure reduces degree-based matching into a b-matching on an augmented graph. Since, however, the ψ functions need only be defined at valid degree inputs, define a concave ψ function as a function satisfying

$$\psi_v(d) - \psi_v(d-1) \quad \geq \quad \psi_v(d+1) - \psi_v(d), \forall d \in \mathbb{N}, 1 \leq d \leq n(v) - 1, \tag{3.2}$$

where $n(v)$ is the original degree $\deg(v, E)$. ***I.e.***, for increasing valid inputs, the change in value of a concave function decreases.

The procedure for solving degree-based matching augments the original graph with auxiliary nodes, where edges between original nodes and auxiliary nodes are set such that their total weights are equivalent to the ψ degree functions, and solves a maximum weight b-matching problem on the augmented graph.

Algorithm 1 Optimization procedure for degree-based matching.

Require: Input weighted graph $G = \{V, E, w\}$, degree preferences $\{\psi_v | v \in V\}$, and maximum weight b-matching solver BMATCHING

Create auxiliary nodes $\tilde{V} = \{\tilde{v}_1, \ldots, \tilde{v}_n\}$
$w_{\text{aug}} \leftarrow w$
$E_{\text{aug}} \leftarrow E$
for $v \in V$ **do**
  for d from 1 to $n(v)$ **do**
    $E_{\text{aug}} \leftarrow E_{\text{aug}} \cup \{(v, \tilde{v}_d)\}$
    $w_{\text{aug}}(v, \tilde{v}_d) \leftarrow \psi_v(d-1) - \psi_v(d)$
  end for
end for
$G_{\text{aug}} \leftarrow \{V \cup \tilde{V}, E_{\text{aug}}, w_{\text{aug}}\}$
$\hat{E}_{\text{aug}} \leftarrow$ BMATCHING(G_{aug})
$\hat{E} \leftarrow \{(u, v) | (u, v) \in \hat{E}_{\text{aug}} \wedge u, v \in V\}$
return $\hat{E}$

The augmented graph G_{aug}, contains a copy of the original graph G as well as a set $\tilde{V}$ of additional auxiliary nodes. An auxiliary node is created for each possible nonzero degree in in the graph, which for node v is all integers in the range $[1, n(v)]$. The augmented graph thus contains at most $|V| - 1$ auxiliary nodes. Each original node $v \in V$ is connected to auxiliary nodes $\{\tilde{v}_d \mid 1 \leq d \leq n(v)\}$ in augmented graph G_{aug}. This construction creates graph $G_{\text{aug}} = \{V_{\text{aug}}, E_{\text{aug}}, w_{\text{aug}}\}$, visualized in Figures 3.2 and 3.3, where

$$\begin{aligned} \tilde{V} &= \{\tilde{v}_1, \ldots, \tilde{v}_n\}, \\ V_{\text{aug}} &= V \cup \tilde{V}, \\ E_{\text{aug}} &= E \cup \{(v, \tilde{v}_d) | 1 \leq d \leq n(v), v \in V\}. \end{aligned}$$

The original edge weights in G_{aug} retain their original values, *i.e.*, for $u \in V$ and $v \in V$, $w_{\text{aug}}(u,v) = w(u,v)$, and the weight between original node v and auxiliary node $\tilde{v}_d \in \tilde{V}$ for $1 \leq d \leq n(v)$ is

$$w_{\text{aug}}(v, \tilde{v}_d) = \psi_v(d-1) - \psi_v(d). \tag{3.3}$$

The weight of the d'th auxiliary edge is the change in preference from degree $d-1$ to degree d. Consequently, while the ψ_v functions have outputs for $\psi_v(0)$, there are no auxiliary nodes labeled $\tilde{v}_0$ associated with that setting (the value $\psi_v(0)$ is used to define the weight of edge $(v, \tilde{v}_1)$). The weights $w_{\text{aug}}(v, \tilde{v}_d)$ are monotonically non-decreasing with respect to the index d due to the concavity of the ψ_v functions. This is seen by substituting the auxiliary weight formula from Equation (3.3) for the concavity definition from Equation 3.2,

$$\begin{aligned} \psi_v(d) - \psi_v(d-1) &\geq \psi_v(d+1) - \psi_v(d) \\ -w_{\text{aug}}(v, \tilde{v}_d) &\geq -w_{\text{aug}}(v, \tilde{v}_{d+1}) \\ w_{\text{aug}}(v, \tilde{v}_d) &\leq w_{\text{aug}}(v, \tilde{v}_{d+1}). \end{aligned}$$

This non-decreasing characteristic is crucial to the correctness of this reduction to b-matching. The construction emulates the optimization from Equation (3.1), which is over edges in G, with an optimization over edges of G_{aug}. The degree constraints in the augmented problem are that each (original) node v must have exactly $n(v)$ neighbors (including any connected auxiliary nodes) and auxiliary nodes have no degree constraints. The augmented problem is the hard-constrained optimization,

$$\hat{E}_{\text{aug}} = \underset{E'_{\text{aug}} \subseteq E_{\text{aug}}}{\operatorname{argmax}} \sum_{(u,v) \in E'_{\text{aug}}} w_{\text{aug}}(u,v) \text{ s.t. } \deg(v, E'_{\text{aug}}) = n(v), \forall v \in V. \tag{3.4}$$

It is important to note that the quantity $n(v)$ in Equation (3.4) is the count of neighbors in the *original* graph. The augmented objective can be conceptualized in the following way: the solver is free to choose any graph structure in the original graph, but is required to maximally select auxiliary edges using its remaining free edges for each node. As the degree of a node with respect to its original neighbors changes, its auxiliary degree must change accordingly. Setting the auxiliary edge weights as described above makes that change equivalent to that of the original degree preference functions. The following theorem proves this equivalence to the original objective.

Theorem 3.1.1. *The total edge weight of the maximum weight subgraph $\hat{E}_{\text{aug}}$ of augmented graph G_{aug}, subject to original nodes having degrees equal to their original neighborhood sizes, differs from the original objective value, which is the total edge weight plus degree preference functions ψ_i, by a fixed additive constant δ.* I.e.*, given graph $G = \{V, E, w\}$, where $w(u, v) \mapsto \mathbb{R}$ for each edge $(u, v) \in E$, and concave degree preference functions $\psi_v(d) \mapsto \mathbb{R}$ for each possible degree d of node $v \in V$, for any induced edge-set $E' \subseteq E$,*

$$\sum_{(u,v)\in E'} w(u,v) + \sum_{v\in V} \psi_v(\deg(v, E')) + \delta = \max_{E'_{\text{aug}} \cap E = E', E'_{\text{aug}} \subseteq E_{\text{aug}}} \sum_{(u,v)\in E'_{\text{aug}}} w_\lambda(u,v) \text{ s.t. } \deg(v, E'_{\text{aug}}) = n(v), \forall v \in V,$$

where δ is a fixed constant independent of E'.

Proof. Consider the edges $E'_{\text{aug}} \cap E$. The weights of possible edges in this intersection are the original weights from function w, and the restriction that $E'_{\text{aug}} \cap E = E'$ means these edge sets are identical and therefore the total weights of w and w_λ over these edges are equal.

What remains is to confirm that the total of the ψ degree preference values agree with the weights of the remaining edges in $\hat{E}_{\text{aug}} \setminus E$, which are edges between original nodes and auxiliary nodes. Recall that the augmented problem's degree constraints require each original node in G_{aug} to have degree equal to the original neighborhood size $n(v)$. By construction, each node v has $2n(v)$ available edges from which to choose: $n(v)$ edges from the original graph and $n(v)$ edges to auxiliary nodes. Moreover, if v selects d original edges, it must maximally select $n(v) - d$ auxiliary edges. Since the auxiliary edges are constructed such that their weights are non-decreasing, the maximum $n(v) - d$ auxiliary edges connect to the last $n(v) - d$ auxiliary nodes, namely auxiliary nodes $\tilde{v}_{d+1}$ through $\tilde{v}_{n(v)}$. Thus, for the Theorem 3.1.1 to hold, the change in a ψ_v preference function value should equal the change in the maximum weight auxiliary edge choice. Formally, the change in weight when node v changes its degree from d to d' is

$$\sum_{j=d+1}^{n(v)} w(v, \tilde{v}_j) - \sum_{j=d'+1}^{n(v)} w(v, \tilde{v}_j) \stackrel{?}{=} \psi_v(d) - \psi_v(d').$$

Terms in the summations cancel to show this equivalence. After substituting the definition of

$w(v_i, \tilde{v}_d)$ from Equation (3.3), the desired equality is revealed with some simple algebra, providing

$$\begin{aligned}
&\sum_{j=d+1}^{n(v)} (\psi_v(j-1) - \psi_v(j)) - \sum_{j=d'+1}^{n(v)} (\psi_v(j-1) - \psi_v(j)) \\
&= \sum_{j=d}^{n(v)} \psi_v(j) - \sum_{j=d+1}^{n(v)} \psi_v(j) - \sum_{j=d'}^{n(v)} \psi_v(j) + \sum_{j=d'+1}^{n(v)} \psi_v(j) \\
&= \psi_v(d) - \psi_v(d').
\end{aligned}$$

This means the original objective value for Equation (3.1) and the weight of the augmented graph change by the same value for different induced subgraphs of G. Hence, the original objective and the total edge weight of the augmented subgraph differ only by a constant δ. □

Since an original node that connects to all of its neighbors connects to no auxiliary edges, the constant difference δ between the augmented graph weight and the original degree preference score is determined by nodes' preference scores for full connectivity. I.e., $\delta = -\sum_{v \in V} \psi_v(n(v))$.

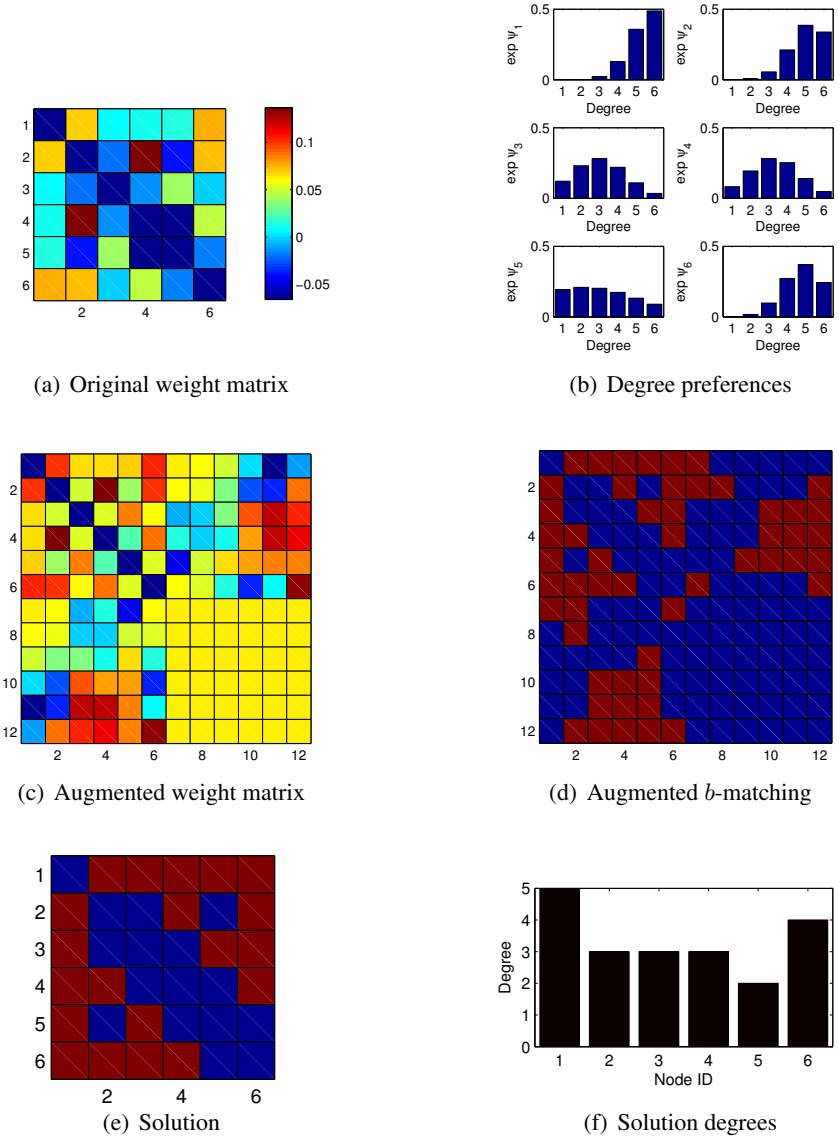
Abstracting a b-matching solver, the algorithm for maximizing objective function (3.1) is summarized in Algorithm 1.

3.1.2 Representing subgraph optimizations with concave degree preferences

The class of problems representable with concave degree preference functions includes k-nearest neighbors, directed and undirected b-matching and degree-constrained subgraph, and ϵ-neighborhood thresholding. This section describes the relationships between these problems and degree-based matching.

First, it is useful to consider how to model directed degree constraints in the problem setting, which is formulated for undirected graphs. Directed problems are represented by duplicating the graph into two bipartitions, each containing copies of the original nodes. One partition represents the out-edges and the other partition represents the in-edges. This graph[4] allows different constraints for in-degree and out-degree, as well as different weights for each direction of each edge.

[4]This duplication construction is related to the so-called *double-cover*, which is a bipartite graph with edges between copies of the original node sets, with edges between each node and the duplicates of each of its original neighbors. This construction differs from a standard double-cover because the asymmetric connectivity between the bipartitions.



(a) Original weight matrix

(b) Degree preferences

(c) Augmented weight matrix

(d) Augmented b-matching

(e) Solution

(f) Solution degrees

Figure 3.3: Example of mapping a degree dependent problem to a hard-constrained b-matching. From left to right, top to bottom: the original weight matrix 3.3(a), original degree preferences 3.3(b), both of which are used to construct the augmented weight matrix 3.3(c), which is pruned to an augmented b-matching 3.3(d). From the augmented b-matching, the solution 3.3(e) is obtained by truncating the auxiliary nodes. The resulting degrees of the nodes 3.3(f) correlate with the original degree preferences.

The k-nearest neighbor subgraph problem finds, for each node, its k nearest neighbors according to some affinity values, which often are computed from a distance metric. The k-nearest neighbor problem differs from b-matching in that k-nearest neighbor graphs are directed; each node greedily connects to its k closest nodes, regardless of how many connections that neighbor already has. Using the duplicated-node construction from above, the k-nearest neighbors degree constraints are representable by concave preference functions which return zero for in-degree k on all nodes and negative-infinity for every other degree. All out-degree preferences are uniform at zero, and the weights of the edges are set to node similarity (or negative distance). Thus, the degree preference functions require in-degree of k but have no influence over out-degree.

Similar degree functions easily encode b-matching in terms of concave preference functions. For example, perfect b-matching constraints are encoded with degree preference functions that again output zero at each node's required degree b_v and negative infinity for all other degrees. More generally, degree-constrained subgraph is encoded by having ψ functions output zero for the range of allowed degrees and negative infinity otherwise.

Finally, ϵ-neighborhood thresholding, in which nodes are connected to all neighbors within distance ϵ, is encoded by a linear penalty on all nodes' degrees. Setting each ψ_v function for all nodes to $\psi_v(d) = -\epsilon d$, the maximum subgraph includes any edge with weight greater than ϵ. This is because the change to the objective value when an edge is added is the weight of the edge plus a single penalty ϵ for increasing the degree by one. Therefore, any edges whose weight is greater than the penalty will add to the objective and will be active at the maximum.

Computing the maximum weight graph under these simple constraints ranges from trivial to solvable with classical approaches, but all of these are generalized by concave preference functions. Thus, applications that use any of these hard-constraints may benefit from generalization to soft degree preferences.

3.1.3 Degree-based matching as maximum likelihood estimation

The degree-based matching objective may be interpreted as a probability distribution over graph structures by exponentiating the objective function. The resulting log-probability distribution is

$$\log \Pr(E') = \sum_{(u,v)\in E'} w(u,v) + \sum_{v\in V} \psi_v(\deg(v, E')) - \log Z(w, \psi),$$

where the partition function Z is

$$Z(w, \psi) = \sum_{E'} \exp\left(\sum_{(u,v) \in E'} w(u, v) + \sum_{v \in V} \psi_v(\deg(v, E')) \right).$$

Computation of this partition function is #P-complete[5], and thus marginal inference is intractable.

Such a probabilistic interpretation provides an intuitive relationship between the combinatorial optimizations subsumed by degree-based matching and statistical problems such as machine learning and prediction.

For example, in the case that the degree preferences are given but the weights are determined by some observed data X, the probability may be interpreted as a *maximum a posteriori* (MAP) estimation. If the observed data associated with each edge is conditionally independent given the presence or non-presence of the edge, the weight is equivalent to the gain in log-likelihood as the edge variable is changed from not present to present,

$$w(u, v) \quad = \quad \log \frac{\Pr(X | (u, v) \in E')}{\Pr(X | (u, v) \notin E')}.$$

Thus, the total likelihood of data X is proportional to the exponentiated weight of all included edges,

$$\Pr(X | E') \quad \propto \quad \exp\left(\sum_{(u,v) \in E'} w(u, v) \right).$$

In this setting, the degree preferences act as a prior probability over edge structures,

$$\Pr(E') \quad \propto \quad \exp\left(\sum_{v \in V} \psi_v(\deg(v, E')) \right).$$

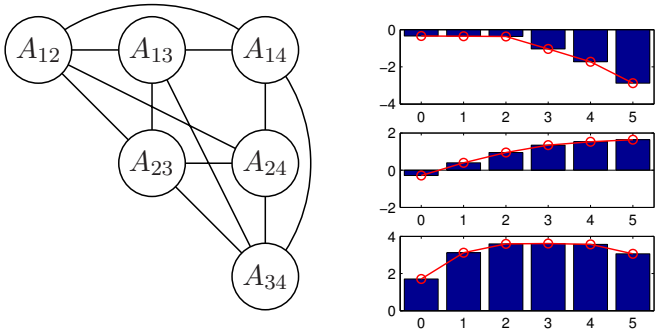
Combining the data likelihood and the edge prior, the resulting optimization is equivalent to the standard MAP inference formulation

$$\hat{E} = \operatorname*{argmax}_{E'} \Pr(X | E') \Pr(E').$$

Using these interpretations may guide model selection in practice. For example, deciding what function to use to determine the weight between nodes may benefit from the log-likelihood ratio

[5] Computing this partition function subsumes the computation of the matrix permanent, which is equivalent to the partition function for a maximum weight matching input problem. The computation of a matrix permanent is #P-complete [Valiant, 1979].

interpretation. Treating the optimization as a MAP inference allows the choice of degree priors to filter the observed data from unlikely measurements.



(a) Dependency graph (b) Valid degree preference functions

Figure 3.4: Left: The dependency graph between edge variables in a four-node graph, where the activation of edge (v_i, v_j) is denoted by its entry A_{ij} in the adjacency matrix. Each edge variable is dependent on the other edges attached to its endpoints, which results in a dense structure. Since we use an undirected adjacency matrix, only the entries in the upper right triangle are depicted in this graphical model. Right: Examples of valid degree preference functions. The preference for each degree is represented by blue bars. The change in the preference value is non-increasing and the piecewise-linear, interpolated curve, represented by the red curve, is a concave function.

Dependency structure among edge variables Since there is a natural probabilistic interpretation of the main objective, it may be tempting to use traditional probabilistic inference techniques to solve the optimization. In the simple case that there are no degree preference functions ψ, the optimization is over independent, Bernoulli variables representing the presence of each edge, and can be solved by simply thresholding the weights at zero, such that all positively weighted edges are included and all negatively weighted edges are excluded. When the node degrees are subject to nontrivial ψ functions, the objective value for each edge becomes dependent on all other edges sharing its endpoints.

Figure 3.4(a) contains an example dependency graph of the edge variables, notated in the binary adjacency matrix form. This densely dependent structure renders standard inference techniques intractable for nontrivial degree functions. Each matrix entry of the triangulated dependency graph

is part of a single, fully connected clique, so exact inference techniques such as the junction tree algorithm require time exponential in the node cardinality of the graph.

3.2 Summary of degree-based subgraph estimation

The optimization problem referred to here as degree-based matching is a general form of degree-based subgraph estimation. The efficient solver described above reduces the problem to a maximum weight b-matching on an augmented graph. Given the reduction, any polynomial-time b-matching solver will find the solution. The next chapter investigates a b-matching solver that uses the probabilistic inference technique *belief propagation*, which passes messages between nodes in the graph in a distributed manner and is guaranteed to find the solution to a well-defined class of maximum weight b-matching problems.

This chapter also discusses the probabilistic interpretation for degree-based subgraph estimation, in which the objective function is interpreted as a log-probability distribution over graph structure. This distribution factorizes into a natural decomposition for graph structures, consisting of potential functions for edge presence and for node degree. For various types of data and settings, these potential functions can be modeled such that maximum likelihood graph structure estimation is useful for machine learning tasks, and Part II discusses some applications of this idea.

Chapter 4

Belief Propagation for b-Matching

In the previous chapter, the problem of degree-based subgraph estimation is reduced to a maximum weight b-matching on an augmented graph. This section details a solver for maximum weight b-matching problems using the approximate probabilistic inference technique known as *belief propagation*, which is proven to compute the exact solution of the b-matching problem. Algebraic manipulations of the standard belief propagation update formulas result in an algorithm with simple, lightweight, parallel update rules that achieve similar asymptotic running time to classical, combinatorial solvers on a class of b-matching problems, and exhibits comparable solution times in practice. Additionally, solving the problem with belief propagation allows heuristic speedups, as well as a naturally parallel algorithm.

4.1 Belief propagation

Belief propagation is an approximate inference technique for *Markov random field* (MRF) representations of probability distributions [Pearl, 1988; Wainwright and Jordan, 2008]. A Markov random field describes a probability distribution function as a product of factors over groups of dependent random variables, where each non-overlapping pair of groups is considered Markov independent, or is independent given the states of all other groups. The technique in this section uses a *pairwise* MRF, in which all groups are of at most two random variables. Belief propagation assigns beliefs for each group and passes messages between the groups to resolve inconsistencies. The two main variants of belief propagation are the *sum-product* algorithm and the *max-product* algorithm. The

sum-product algorithm aims to compute the *marginal probabilities* of each variable group, which are the total joint likelihood of each variable state. In contrast, the max-product algorithm aims to compute the *max-marginals* of each group, which are the maximum possible joint likelihood given each state of the group's variables. This section primarily focuses on the max-product algorithm, but a discussion of the sum-product algorithm on the Markov random field for maximum-weight matching, in which the estimated marginals can be used to approximate the *matrix permanent*, is in Appendix A.1.

An alternate probabilistic framework to Markov random fields is the *factor graph* model, in which probability distributions are expressed as a bipartite graph between variable nodes and factor nodes. Using the factor graph representation, an alternate derivation of belief propagation for maximum weight b-matching is available with different intuitions yet the same final resulting algorithm. The factor graph formulation has been used to implement the *affinity propagation* algorithm [Frey and Dueck, 2006], where reasoning with high-order factor potentials allows a similar simplification of message passing as in the main text of this thesis [Givoni and Frey, 2009; Tarlow *et al.*, 2010].

4.1.1 Related work

This section discusses the use of *belief propagation* for solving maximum weight b-matchings [Huang and Jebara, 2007; Huang and Jebara, 2011]. Belief propagation [Pearl, 1988] is essentially a dynamic programming approach to computing the marginals and max-marginals of a tree-structured Markov random field. The application in this chapter uses the form of belief propagation known as *max-product*, which finds the most likely state of the variables. Belief propagation is used extensively on cyclic graphical models in practice, despite the fact that neither convergence nor correctness of the resulting marginals is guaranteed in general. Theoretical guarantees on the performance of belief propagation on certain classes of loopy graphical models are thus important. In a graphical model with a single loop, max-product belief propagation converges and yields the true solution [Weiss, 2000]. More recently, max-product was proved to converge on a graphical model representing bipartite maximum weight matching in a bounded number of iterations [Bayati *et al.*, 2005; Bayati *et al.*, 2008]. A similar analysis extends this result to bipartite b-matchings [Huang and Jebara, 2007]. In any matching graph, tightness of the *linear programming* (LP) relaxation is a nec-

essary and sufficient condition for guaranteed convergence of max-product [Sanghavi *et al.*, 2007; Wainwright and Jordan, 2008]. While combinatorial algorithms such as balanced network flow [Fremuth-Paeger and Jungnickel, 1999] solve maximum weight b-matching, the belief propagation approaches provide lightweight algorithms with simple update formulas that are easily run in parallel. Relatedly, belief propagation has been recently proven to converge on min-cost flow problems, which can be used to solve b-matching problems [Gamarnik *et al.*, 2010].

Belief propagation has been shown in some cases to have theoretically optimal running time [Salez and Shah, 2009]. Belief propagation has a constant iteration count for convergence on a bipartite 1-matching on graphs where the edge costs are drawn independently and identically distributed (*iid*) from a light-tailed distribution. Under a light-tailed distribution, large weights are unlikely, allowing that for any error threshold $\epsilon > 0$, there exists a number of iterations $h(\epsilon)$ and graph size $N(\epsilon)$ such that after $h(\epsilon)$ iterations of belief propagation, the fraction of suboptimal assignments is less than ϵ. For graphs of size greater than $N(\epsilon)$, the number of iterations to the convergence threshold does not depend on N. Since each iteration costs $\mathcal{O}(N^2)$, the total expected cost of solving a bipartite 1-matching is then $\mathcal{O}(N^2)$ quadratic, which is also the time required to read the input.

Faster standard max-product updates are possible by exploiting repeated potentials, which can be sorted in advance of message updates [McAuley and Caetano, 2010]. Each message update, which involves a maximization over potential sums, can use the sorted orders such that the expected running time of each update for variables with N settings is $\mathcal{O}(\sqrt{N})$. This speedup is particularly useful when variables have many possible settings or factors are high-order, which is the case in the generalized matching function.

4.1.2 Maximum weight matching solvers

Maximum weight matching is one of the most studied combinatorial optimizations in graph theory. Maximum weight matching in a bipartite graph is equivalent to the *assignment problem*. Classical approaches to finding the maximum weight bipartite matching include the *augmenting path algorithm* [Jonker and Volgenant, 1979], and the Hungarian algorithm [Munkres, 1957], as well as converting the problem to a minimum-cost maximum flow problem [Goldberg and Kennedy, 1995]. The non-bipartite matching problem is significantly harder than its bipartite form, and requires ma-

nipulation of a combinatorial number of structures known as *blossoms*, which are odd-length cycles [Edmonds, 1965]. Nevertheless, clever bookkeeping allows for efficient solution of non-bipartite problems.

Various advances on fast solvers for maximum weight matching have been proposed. For graphs restricted to nonnegative integer weights, the bipartite maximum weight 1-matching problem was shown to be solvable in $\mathcal{O}(\sqrt{|V|}|E|\log(|V|))$ time [Gabow and Tarjan, 1989]. Recently, an $\tilde{\mathcal{O}}(|V|^{2.376})$ randomized algorithm for integer weights which succeeds with high probability was revealed [Sankowski, 2009]. Subsequently, a (1-ϵ) approximation algorithm for nonbipartite maximum weight matching with real weights runs in $\mathcal{O}(|E|\epsilon^{-2}\log^3|V|)$ time [Duan and Pettie, 2010].

4.1.3 The max-product algorithm

In a *pairwise Markov random field* over variables $\{x_1, \ldots, x_n\}$, the log-probability of any state is [6]

$$\log \Pr(X) \quad = \quad \sum_i \Phi_i(x_i) + \sum_{ij} \Psi_{ij}(x_i, x_j) - \log Z(\Phi, \Psi),$$

where $Z(\Phi, \Psi)$ is the *normalizing partition function*. The standard max-product update rules maintain a message vector m_{ij} from each variable x_i to variable x_j,

$$m_{ij}^t(x_j) \quad = \quad \max_{x_i} \left[\Phi_i(x_i) + \Psi_{ij}(x_i, x_j) + \sum_{k \neq j} m_{ki}^{t-1}(x_i) \right],$$

which combines incoming messages from all nodes except the message receiver with its own local potentials to form outgoing messages. Given all incoming messages, the belief at iteration t for variable state x_i combines all incoming messages with the local potential,

$$B^t(x_i) \quad = \quad \Phi(x_i) + \sum_j m_{ji}^{t-1}(x_i).$$

The remainder of this chapter derives the MRF structure and its potential functions for representing the maximum weight b-matching problem, and describes the algebraic simplifications that provide scalability improvements.

[6] In this section, we restrict the MRFs to be pairwise, so we explicitly write the pairwise potentials ψ and singleton potentials ϕ. In general, the formula may be a sum over all groups, where the groups may be of any size.

4.2 Fast belief propagation for maximum weight b-matching

The structure of the maximum weight b-matching problem lends itself to a natural representation as a *Markov random field*, in which random variables represent the matching assignments and functions of these random variables represent both the constraints and the weights. Representing the problem as a probabilistic system means probabilistic inference techniques, which recover the most likely setting of the random variables, recover the maximum weight b-matching. This section first describes the probabilistic representation of a maximum weight b-matching, the inference technique and convergence guarantees, and the simplification steps necessary to reduce the combinatorial complexity of the default representation, as well as some additional techniques to further reduce computational complexity in the expected case. The derivations in this section provide the formulas for perfect b-matchings, but non-perfect b-matchings are also possible [Sanghavi *et al.*, 2007].

In a node-based representation of b-matching [Huang and Jebara, 2007], the b-matching is represented by variables $\{x_v | v \in V\}$, the values of which represent the neighbors of v in the b-matching $\hat{E}$. For example, in Figure 3.1, node v_1 is matched with nodes v_7 and v_8, so the value of x_{v_1} encodes the set $\{v_7, v_8\}$. Let function S convert a node variable to its encoded neighbor-set, *e.g.*, $S(x_{v_1}) = \{v_7, v_8\}$. Since node v must have degree b_v, x_v has $\binom{n(v)}{b_v}$ possible settings, each representing a possible set of b_v neighbors. Since likelihoods of independent events are multiplicative, the additive weights of a maximum weight b-matching problem are represented as log-potentials:

$$\Phi_u(x_u) \quad = \quad \sum_{v \in S(x_u)} w(u, v). \tag{4.1}$$

These weight potentials are tied together by pairwise b-matching compatibility functions, which ensure the agreement between all nodes' variable states.

$$\Psi_{uv}(x_u, x_v) \quad = \quad \begin{cases} 0 & \text{if both variables indicate connectivity } (v \in S(x_u)) \wedge (u \in S(x_v)) \\ 0 & \text{if neither variable indicates connectivity } (v \notin S(x_u)) \wedge (u \notin S(x_v)) \\ -\infty & \text{otherwise } (x_u \text{ and } x_v \text{ disagree}). \end{cases}$$

Using the defined potential functions Φ and the pairwise compatibility functions Ψ, the maximum weight b-matching objective is equivalent finding the most likely state of variable set $\{x_u | u \in V\}$

for unnormalized log-likelihood

$$\sum_{u \in V} \Phi_u(x_u) + \sum_{u,v \in V} \Psi_{uv}(x_u, x_v).$$

The standard max-product algorithm iteratively passes messages vectors between variables and computes beliefs, which are estimates of max-marginals. The standard update equations for beliefs B and messages $m_u(x_v)$ from variable x_u to x_v are

$$\begin{aligned} m_{uv}^t(x_v) &= \max_{x_u} \left[\Phi_u(x_u) + \Psi_{uv}(x_u, x_v) + \sum_{a \in V \setminus v} m_{au}^{t-1}(x_u) \right], \\ B^t(x_u) &= \Phi_u(x_u) + \sum_{v \in V} m_{vu}^{t-1}(x_u). \end{aligned}$$

To alleviate some notational clutter, define the message from a node to itself as a vector of all zeros. Section 4.2.2 and Section 4.2.3 include algebraic and algorithmic methods for storing and computing these messages exactly and efficiently, but first, Section 4.2.1 establishes that max-product converges to the correct solution in a number of iterations linear in the graph size.

4.2.1 Convergence analysis

In general, a cyclic graphical model like the one described above suffers from two possible problems: the messages may never converge, and they may converge to a state that is not the global optimum. This section presents a theorem that guarantees neither of these will occur in bipartite graphs; the algorithm will converge to the true optimum in a bounded number of iterations. The convergence guarantee is then extended to apply toward maximum weight b-matchings on non-bipartite graphs whose LP relaxations are tight.

The bipartite convergence guarantee requires two preconditions: first, that the edge weights are bounded, and, second, that the optimal b-matching is unique. Let weight function w be overloaded such that, given an input edge set, it outputs the sum of edge weights, $w(E) = \sum_{e \in E} w(e)$. For input graph $G = \{V, E, w, b\}$, let $M(G)$ be the set of edge sets corresponding to valid b-matchings.

The analysis considers the belief state of a particular node after some iteration of belief propagation. The max-product belief of a node in a loopy graphical model after iteration t is known to be equivalent to the max-marginal of the node's *unwrapped graph* [Weiss, 2000]. The unwrapped graph is equivalent to the computation tree of messages passed during belief propagation, and

is constructed by following the message propagation in reverse. Formally, the unwrapped graph $T_v = \{V^T, E^T\}$ of node $v \in V$ is rooted by a copy of v, denoted r. We denote this copying relationship with mapping function τ, which maps nodes in V^T to their corresponding nodes in V. *E.g.*, $\tau(r) = v$. The children of the root are copies of v's neighbors in G. Then, the children of each interior node $u \in V^T$ are the neighbors of $\tau(u)$ in G except the node corresponding to u's parent. The paths in an unwrapped graph of height h represent all possible non-backtracking walks of length h in G starting from root node v. The non-backtracking, unwrapped graph construction is equivalent to the similarly non-backtracking message update rule, such that messages propagated from the leaves of the unwrapped graph to the root are the same messages passed during loopy belief propagation. Since, however, the unwrapped graph is a tree and contains no cycles, the beliefs for the root node after propagating messages upwards are the true max-marginals on the tree. For the b-matching graphical model, this corresponds to a *near-perfect* maximum weight b-matching on the tree, where all nodes but the leaf nodes have their exact target degree.

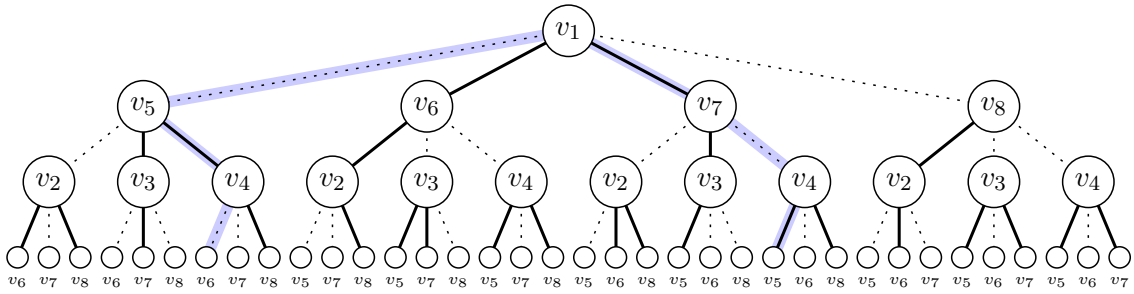


Figure 4.1: Example of an unwrapped graph at three iterations. A tree b-matching is highlighted based on the b-matching from Figure 3.1. Note that leaf nodes cannot have perfect b-matchings, but all inner nodes and the root do. One possible alternating path is highlighted, which is discussed in the convergence proof.

Using the unwrapped graph construction, the following two quantities are equivalent: the number of iterations of loopy belief propagation on G such that node v's belief for the optimal b-matching is guaranteed to exceed the belief for any suboptimal b-matching, and the height of tree T_v such that the maximum weight b-matching on T_v connects root node r to the corresponding

neighbors of v in the maximum weight b-matching of G. Thus, the following theorem bounding the convergence time of belief propagation is proved by analysis of the unwrapped graph in the Appendix B.

Theorem 4.2.1. *Given bipartite input graph $G = \{V, E, w, b\}$, such that a unique, optimal b-matching $\hat{E}$ has weight greater by constant ϵ than any other b-matching,* i.e.*,*

$$\epsilon \leq w(\hat{E}) - \max_{E' \in M, E' \neq \hat{E}} w(E'),$$

and all weights are bounded inclusively between $w_{\min}$ and $w_{\max}$, i.e.*,*

$$w_{\min} \leq w(e) \leq w_{\max}, \forall e \in E,$$

the maximum belief at iteration t for any node $v \in V$ indicates the corresponding neighbors in true optimum $\hat{E}$ when $t = \Omega(|V|)$.

The convergence conditions are further relaxed to include any maximum weight b-matching problem for which the linear programming relaxation is tight [Bayati *et al.*, 2007; Bayati *et al.*, To appear], as claimed in the following theorem

Theorem 4.2.2 (Theorem 3.1 of [Bayati *et al.*, To appear])**.** *Assume that the linear programming relaxation of the b-matching problem on G has an integer solution and a unique optimum, and that the weights in G are bounded. Then belief propagation converges to the optimal solution in $\mathcal{O}(|V|)$ iterations.*

The proof for Theorem 4.2.2, including rigorous definitions of the LP relaxation is in Appendix B.

In both the theoretical analysis and our empirical observations, the running time is dependent on the ϵ value for the input, and thus, the algorithm is not strongly polynomial. This dependence causes two scenarios on which belief propagation fails in practice. The first is when ϵ is zero, which happens when at least two b-matchings achieve the maximum weight, and the second is when ϵ is very small, in which case the number of iterations necessary for convergence is very large. In theory, adding random noise to the weights corrects the first scenario, but in practice, this tends to create the second scenario, in which the artificially nonzero ϵ is quite small. While typical ϵ's are reasonably large enough for fast convergence in many applications, techniques for handling small ϵ problems are the subject of future work.

4.2.2 Message simplification

In this section, a simplified update rule is derived which compactly and exactly computes the standard max-product message updates by exploiting the fixed structure of the matching compatibility functions. In the standard max-product algorithm, each message between variables is a vector, with an entry representing each possible setting of the receiver variable. In the b-matching model, however, each entry of these message-vectors is always one of two possible values, which, since the message vectors are scale-invariant, can be summarized as a ratio of the two values. Thus, with careful bookkeeping, all messages can be compressed to single belief values for each candidate edge. Updating an edge's belief-value is possible in time proportional to the number of candidate neighbors for the source node, again by exploiting the structure of the matching problem to circumvent the combinatorially large set of possible states for the source node's variable. The edge-based beliefs can then be further summarized per node by unrolling one level of message-passing recursion, resulting in only $\mathcal{O}(|V|)$ additional storage during belief propagation.

We exploit three peculiarities of the above formulation to fully represent the $\binom{n(v)}{b_i}$ length messages as scalars. First, the Ψ_{uv} functions are well structured, and their structure causes the maximization term in the message updates to always be one of two values:

$$m_{uv}(x_v) \quad = \quad \begin{cases} \max_{x_u | v \in S(x_u)} \Phi_u(x_u) + \sum_{a | a \in V \setminus v} m_{au}(x_u) & \text{if } u \in S(x_v) \\ \max_{x_u | v \notin S(x_u)} \Phi_u(x_u) + \sum_{a | a \in V \setminus v} m_{au}(x_u) & \text{if } u \notin S(x_v) \end{cases}.$$

The reason there are only two possible values is because the Ψ_{uv} function changes based only on whether the setting of x_v indicates that v shares an edge with u. Furthermore, if we redefine the above message values as two scalars, we can write the messages as

$$\begin{aligned} \mu_{uv} &= \max_{x_u | v \in S(x_u)} \Phi_u(x_u) + \sum_{a \in S(x_u) \setminus v} \mu_{au} + \sum_{a \notin S(x_u) \setminus v} \nu_{au}, \\ \nu_{uv} &= \max_{x_u | v \notin S(x_u)} \Phi_u(x_u) + \sum_{a \in S(x_u) \setminus v} \mu_{au} + \sum_{a \notin S(x_u) \setminus v} \nu_{au}, \end{aligned}$$

Second, since the messages are derived from unnormalized log-probabilities, subtracting any constant from the vectors does not change the result. Subtract ν_{uv} from all entries in the message vector to get

$$\hat{\mu}_{uv} = \mu_{uv} - \nu_{uv} \quad \text{and} \quad \hat{\nu}_{uv} = 0.$$

This lossless compression scheme simplifies the storage of message vectors from length $\binom{n(v)}{b_u}$ to a single scalar per message,

$$\hat{\mu}_{uv} = \underbrace{\left(\max_{v \in S(x_u)} \Phi_u(x_u) + \sum_{a \in S(x_u) \setminus v} \hat{\mu}_{au} \right)}_{\text{Part1}} - \underbrace{\left(\max_{v \notin S(x_u) \setminus v} \Phi_u(x_u) + \sum_{a \in S(x_u) \setminus v} \hat{\mu}_{au} \right)}_{\text{Part2}} \quad (4.2)$$

The third exploitable peculiarity of these message updates is that the new scalar representation of the messages $\hat{\mu}_{uv}$, decompose such that the sums in the positive component and the sums in the negative component are so similar that most of the computation cancels out. Expanding the variable potentials Φ inside each maximization, the first maximization (part 1) takes the form

$$\max_{v \in S(x_u)} \sum_{a \in S(x_u)} w(a,u) + \sum_{a \in S(x_u) \setminus v} \hat{\mu}_{au}.$$

The constraint on the maximization requires that node the maximizing state x_u connects u to v, so the expression above is equivalently written as

$$w(u,v) + \max_{v \in S(x_u)} \sum_{a \in S(x_u) \setminus v} (w(a,u) + \hat{\mu}_{au}).$$

The second term (part 2) of Equation (4.2) similarly decomposes, except the maximization constraint requires that the state x_u does not connect u to v, resulting in the expression

$$\max_{v \notin S(x_u) \setminus v} \sum_{a \in S(x_u)} w(a,u) + \sum_{a \in S(x_u) \setminus v} \hat{\mu}_{au} = \max_{v \notin S(x_u) \setminus v} \sum_{a \in S(x_u)} (w(a,u) + \hat{\mu}_{au})$$

Recombining the two expressions, the message update is

$$\hat{\mu}_{uv} = w(u,v) + \max_{v \in S(x_u)} \sum_{a \in S(x_u) \setminus v} (w(a,u) + \hat{\mu}_{au}) - \max_{v \notin S(x_u) \setminus v} \sum_{a \in S(x_u)} (w(a,u) + \hat{\mu}_{au})$$

The terms inside the summations cancel, since the first component contains the $b_u - 1$ greatest values of $w(a,u) + \hat{\mu}_{au}$, and the second component contains the b_u greatest values, leaving only the negation of the b_u'th greatest entry. The *selection* operation finds the k'th largest element of a set for some index k. For notational convenience, denote the selection operation over any set S as

$$\sigma_k(S) = s \in S \text{ where } |\{t \in S | t \geq s\}| = k.$$

Then the simplified message update is

$$\hat{\mu}_{uv} = w(u,v) - \sigma_{b_u}(\{w(a,u) + \hat{\mu}_{au} | a \in V \setminus v\}). \quad (4.3)$$

A final algebraic manipulation allows elimination of the messages altogether, resulting in a direct belief update rule. Using the current simplified message from Equation (4.3), the belief for any state is

$$B^t(x_u) \quad = \quad \sum_{v \in S(x_u)} w(u,v) + \hat{\mu}^t_{vu},$$

which is a purely additive combinatorial sum, and thus can be maximized by greedily choosing the b_u greatest values in $\{w(u,v) + \hat{\mu}_{uv} | v \in V \setminus u\}$. Each of these sums acts as an edge-wise belief

$$B^t_{uv} = w(u,v) + \hat{\mu}^t_{vu}.$$

Substituting the message update rule (4.3), the following simplifications are available

$$B^t_{uv} = w(u,v) + (w(v,u) - \sigma_{b_v}(\{w(a,v) + \hat{\mu}_{av} | a \in V \setminus u\}))$$

Since the problem parameterization handles an undirected graph, weights $w(u,v)$ and $w(v,u)$ are equal, and a constant doubling can be dropped. Furthermore, replacing weight $w(a,v)$ with $w(v,a)$ inside the selection makes the selection over edge-wise beliefs themselves, leaving the simplified belief update rule

$$B^t_{uv} = w(u,v) - \sigma_{b_v}(\{B^{t-1}_{va} | a \in V \setminus u\}). \tag{4.4}$$

Additionally to the simplifications provided so far, the update rule above still contains enough structure to allow even further scalability improvements, described in the remainder of this section and the next section.

Memory savings by further unrolling recursion By unrolling the recursion of the belief update rule, storing full beliefs becomes unnecessary. Instead, all that must be stored are the *selected* beliefs, because the selection operation in Equation (4.4) only weakly depends on sender node u. That is, the selection operation is over all nodes except u, which means the selected value will be either the b_v'th or the $(b_v + 1)$'th greatest element,

$$\sigma_{b_v}(\{B^{t-1}_{va} | a \in V \setminus u\}) \in \qquad \{\sigma_{b_v}(\{B^{t-1}_{va} | a \in V\}), \sigma_{b_v+1}(\{B^{t-1}_{va} | a \in V\})\}.$$

Thus, once each row of the belief matrix $\mathbf{B}$ is updated, these two selected values can be computed and stored, and the rest of the row can be deleted from memory. Any further reference to $\mathbf{B}$ is

therefore abstract, as it is never instantiated in practice. Entries of the belief matrix can be computed in an online manner from the stored selected value. Let α_v be the negation of the b_v'th selection and β_v be that of the $(b_v + 1)$'th selection. Then the update rules for these parameters are

$$\alpha_v^t = -\sigma_{b_v}(\{B_{va}^{t-1} | a \in V\}), \;\; \beta_v^t = -\sigma_{b_v+1}(\{B_{va}^{t-1} | a \in V\}), \tag{4.5}$$

and the resulting belief lookup rule is

$$B_{uv}^t = w(u,v) + \begin{cases} \alpha_v^t & \text{if } (v,u) \notin \hat{E}^{t-1} \\ \beta_v^t & \text{otherwise.} \end{cases}$$

At the end of each iteration, the current estimate of $\hat{E}$ is

$$\hat{E}^t = \{(u,v) | B_{uv}^{t-1} \geq \alpha_u^t\},$$

which is computed when the α and β values are updated in Equation (4.5). When this estimate is a valid b-matching, *i.e.*, when all nodes have their target degrees, the algorithm has converged to the solution. The algorithm can be viewed as simply computing each row of the belief matrix and performing the selections on that row and is summarized in Algorithm 2.

Using the simplifications described in this section, the running time per iteration is primarily dependent on the time to perform the selection operation. In the worst case, the best known algorithm is $\mathcal{O}(N)$ time to perform select for any k, however the worst-case linear time algorithm is slower in practice than the expected linear time algorithm QUICKSELECT [Cormen *et al.*, 2001], which is a QUICKSORT-like recursive partitioning algorithm. In the next section, an even faster algorithm is described that exploits the structure of the selection operations necessary in this belief propagation algorithm.

4.2.3 Fast message updates via sufficient selection

This section describes the running time enhancement in the proposed algorithm, which is a variation of the faster belief propagation algorithm proposed by [McAuley and Caetano, 2010]. The enhancements aim to reduce the running time of each iteration by exploiting the nature of the quantities being selected. In particular, the key observation is that each belief is a sum of two quantities: a weight and an α or β value. These quantities can be sorted in advance, outside of the inner (row-wise) loop of the algorithm, and the selection operation can be performed without searching over the

Algorithm 2 Belief propagation for b-matching. Computes the maximum weight b-matching.

$\alpha_v^0 \leftarrow 0, \forall v \in V$

$\beta_v^0 \leftarrow 0, \forall v \in V$

$\hat{E}^0 = \emptyset$

$t \leftarrow 1$

while not converged **do**

  for all $u \in V$ **do**

    $\alpha_u^t \leftarrow -\sigma_{b_u}(\{B_{ua}^{t-1} | a \in V\})$ {e.g., Algorithm 3}

    $\beta_u^t \leftarrow -\sigma_{b_u+1}(\{B_{ua}^{t-1} | a \in V\})$

    $\hat{E}^t = \{(u,v) | B_{uv}^{t-1} \geq \alpha_u^t\}$

  end for

  $\hat{E}^t = \{(u,v) | B_{uv}^{t-1} \geq \alpha_u^t\}$

  delete α^{t-1} and β^{t-1} from memory

  $t \leftarrow t + 1$

end while

entire row, significantly reducing the amount of work necessary. This is done by testing a stopping criterion that guarantees no further belief lookups are necessary.

Some minor difficulties arise, however, when sorting each component, so the algorithm by [McAuley and Caetano, 2010] does not directly apply as-is. First, the weights cannot always be fully sorted. In general, storing full order information for each weight between all pairs of nodes requires quadratic space, which is too expensive with larger data sets. Thus, the proposed algorithm instead stores a cache of the heaviest weights for each node. In some special cases, such as when the weights are a function of Euclidean distance, data structures such as *kd-trees* can be used to implicitly store the sorted weights [Bentley, 1975], or various data structures can compute approximate nearest-neighbors [Gionis *et al.*, 1999; Maneewongvatana and Mount, 1999]. Such constructions can provide one possible variant to our main algorithm.

Second, the α-β values require careful sorting, because the true belief updates mostly include α^t terms but a few β^t terms. Specifically, the indices that index the greatest b_v elements of the row should use β^t. One way to handle this technicality is to first compute the sort-order of the α^t terms and, on each row, correct the ordering using a binary search-like strategy for each index in the

selected indices. This method is technically a logarithmic time procedure, but requires some extra indexing logic that creates undesirable constant time penalties. Another approach, which is much simpler to implement and does not require extra indexing logic, is to use the sort-order of the β^t's and adjust the stopping criterion to account for the possibility of unseen α^t values.

Since the weights do not change during belief propagation, at initialization, the algorithm computes index cache $\mathbf{I} \in \mathbb{N}^{|V| \times c}$ of cache size c, which is a parameter set by the user, where entry I_{uk} is a pointer to the k'th largest weight neighbor connected to node x_u and, for $v = I_{uk}$,

$$w(u, v) = \sigma_k(\{w(u, a) | a \in V\}).$$

At the end of each iteration, the β^t values are similarly sorted and stored in index vector $\mathbf{e} \in \mathbb{N}^{|V|}$, where, for $v = e_k$, entry $\beta_v^t = \sigma_k(\beta_a^t | a \in V\})$.

The selection operation from (4.5) is then computed by checking the beliefs corresponding to the sorted weight and β indices. At each step, maintain a set S of the greatest $b_v + 1$ beliefs seen so far. These provide tight lower bounds on the true $\alpha - \beta$ values. At each stage of this procedure, the current estimates for α_v^t and β_v^t are

$$\tilde{\alpha}_v^t \leftarrow \sigma_{b_v}(S), \text{ and } \tilde{\beta}_v^t \leftarrow \min(S).$$

Incrementally scan the beliefs for both index lists $(\mathbf{I})_v$ and $\mathbf{e}$, computing for incrementing index k, $B_{u,I_{uk}}$ and B_{u,e_k}. Each of these computed beliefs is compared to the beliefs in set S and if any member of S is less than the new belief, the new belief replaces the minimum value in S.[7]). This maintains S as the set of the greatest $b_v + 1$ elements seen so far.

At each stage, the greatest possible unseen belief is bounded by the sum of the least weight seen so far from the sorted weight cache and the least β value so far from the β cache. Once the estimate $\tilde{\beta}_v^t$ is less than or equal to this sum, the algorithm can exit because further comparisons are unnecessary. Algorithm 3 summarizes the sufficient selection procedure, and Figure 4.2 visualizes the selection process.

The theorem below verifies that the bound from the sufficient selection procedure holds even

[7] A small hash table for the indices will indicate whether an index has been previously visited in $\mathcal{O}(1)$ time per lookup. Furthermore, for small values of b_v where ($b_v << |V|$), a linear scan through S to find the minimum is sufficiently fast, but a priority queue can be used to achieve sub-linear time insertion and replacement when b_v is large.

Algorithm 3 Sufficient selection for belief propagation b-matching. Given sort-order of β^t values and partial sort-order of weights, selects the b_u'th and $b_u + 1$'th greatest beliefs of node u.

$k \leftarrow 1$
bound $\leftarrow \infty$
$S \leftarrow \emptyset$
$\tilde{\alpha}_u^t \leftarrow -\infty$
$\tilde{\beta}_u^t \leftarrow -\infty$
while $\tilde{\beta}_u^t <$ bound **do**
  if $k \leq c$ **then**
    $v \leftarrow I_{uk}$
    if (v is unvisited and $(B_{uv}^{t-1} > \min(S))$ **then**
      $S \leftarrow (S \setminus \min(S)) \cup B_{uv}^{t-1}$
    end if
  end if
  $a \leftarrow e_k$
  if (v is unvisited and $(B_{ua}^{t-1} > \min(S))$ **then**
    $S \leftarrow (S \setminus \min(S)) \cup B_{ua}^{t-1}$
  end if
  bound $\leftarrow w(u, v) + \beta_a^{t-1}$
  $\tilde{\alpha}_u^t \leftarrow \sigma_{b_u}(S)$
  $\tilde{\beta}_u^t \leftarrow \sigma_{b_u+1}(S)$
  $k \leftarrow k + 1$
end while
$\alpha_u^t \leftarrow -\tilde{\alpha}_u^t$
$\beta_u^t \leftarrow -\tilde{\beta}_u^t$

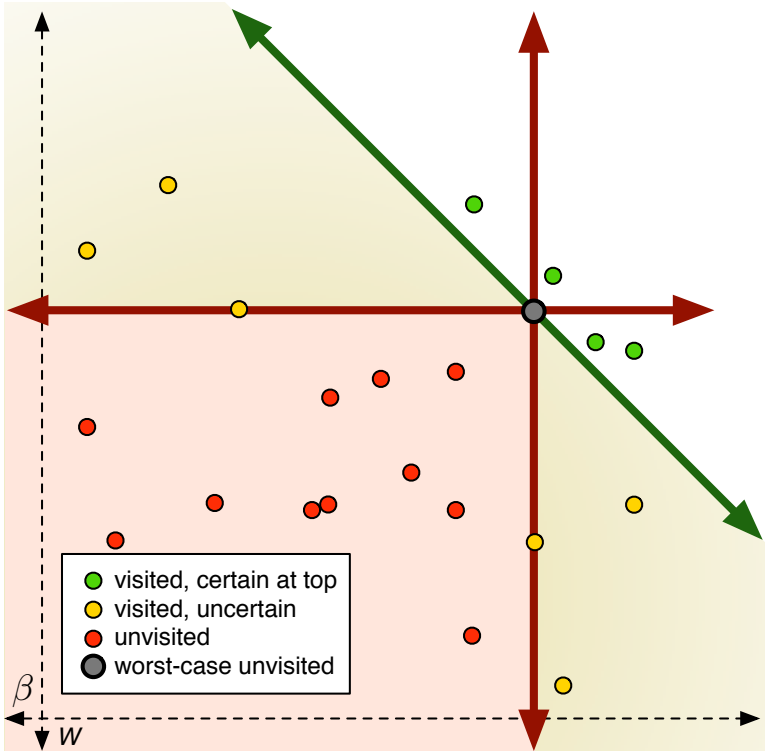


Figure 4.2: Visualization of the sufficient selection process. All points to the right of the vertical red line and above the horizontal red line have been visited, all points in the bottom left quadrant are unknown, but none of them can have greater magnitude than the corner gray point. Thus, the green points with greater magnitude than the gray, worst-case unseen point are certainly the greatest-magnitude points.

though it is computed using only the β values, when many of the beliefs are actually computed using α values.

Theorem 4.2.3. *At each stage of the scan, where set S contains the $b_u + 1$ greatest beliefs corresponding to the first through k'th indices of $(\mathbf{I})_u$ and $\mathbf{e}$, the following properties are invariant: the current estimates bound the true values from below, $\tilde{\alpha}_u^t \leq \alpha_u^t$, $\tilde{\beta}_u^t \leq \beta_u^t$, and the greatest unexplored belief is no greater than the sum of the least cached weight and the least β_u^{t-1} value,*

$$w(u,v) + \beta_a^{t-1} \geq \max\left(\left\{B_{u\ell}^{t-1} | \ell \in \{e_{k+1}, \ldots, e_{m+n}\}\right\}\right), \tag{4.6}$$

where $u = I_{jk}$ and $v = e_k$.

Proof. The first two inequalities follow from the fact that the algorithm is selecting from but has not necessarily seen the full row yet. The third inequality (4.6) is the result of two bounds. First, the beliefs in the right-hand side can be expanded and bounded by ignoring the conditional in the belief update rule and always using β_ℓ^{t-1}:

$$W(x_j, x_\ell) + \beta_\ell^{t-1} \geq B_{j\ell}^{t-1}.$$

By definition $\alpha_\ell^{t-1} \leq \beta_\ell^{t-1}$, since the former is the negation of a larger value than the latter. A sufficient condition to guarantee Inequality (4.6) is then

$$W(x_j, x_u) + \beta_v^{t-1} \geq \max(\{W(x_j, x_\ell) + \beta_\ell^{t-1} | \ell\}),$$

where ℓ is in the remaining unseen indices as in (4.6). Since each component on the left-hand side has been explored in decreasing order, the maximization on the right can be broken into independent maximizations over each component, and neither can exceed the corresponding value on the left. □

Thus, the algorithm will never stop too early. However, the running time of the selection operation depends on how early the stopping criterion is detected. In the worst case, the process examines every entry of the row, with some overhead checking for repeat comparisons. For random orderings of each dimension (and no truncated cache size), the expected number of belief comparisons necessary is $\mathcal{O}(\sqrt{N})$ to find the maximum, where, in our case $N = m + n = |V|$ [McAuley and Caetano, 2009; McAuley and Caetano, 2010]. We show that selection is computable with $\mathcal{O}(\sqrt{bN})$ expected comparisons. However, for problems where the orderings of each dimension are negatively correlated, the running time can be worse. In the case of b-matching, the orderings of the beliefs and potentials are in fact negatively correlated, but in a weak manner. We first establish the expected performance of the sufficient selection algorithm under the assumption of randomly ordered β values.

Theorem 4.2.4. *Considering the element-wise sum of two real-valued vectors $\vec{w}$ and $\vec{\beta}$ of length N with independently random sort orders, the expected number of elements that must be compared to compute the selection of the b'th greatest entry $\sigma_b(\{w_i + \beta_i | i\})$ is $\sqrt{bN}$.*

Proof. The sufficient selection algorithm can be equivalently viewed as checking element-wise sums in the sort orders of the $\vec{w}$ and $\vec{\beta}$ vectors, and growing a set of k indices that have been examined. The algorithm can stop once it has seen b entries that are in the first k of both sort orders.

We first consider the algorithm once it has examined k indices of each vector, and derive the expected number of entries that will be in both sets of k greatest entries. Since the sort orders of each set are random, the problem can be posed as a simple sampling scenario. Without loss of generality, consider the set of indices that correspond to the greatest k entries in $\vec{w}$. Examining the greatest k elements of $\vec{\beta}$ is then equivalent to randomly sampling k indices from 1 to N without replacement. Thus, the probability of any of the k greatest entries of $\vec{\beta}$ being sampled is k/N, and, since there are k of these, the expected number of sampled entries that are in the greatest k entries of both vectors is k^2/N.

Finally, to determine the number of entries the algorithm must examine to have, in expectation, b entries in the top k, we simply solve the equation $b = k^2/N$ for k, which yields that when $k = \sqrt{bN}$, the algorithm will in the expected case observe b entries in the top k of both lists and therefore completes computation. □

Applying the estimated running time to analysis of the full matching algorithm, the following arguments describe the scaling behavior of the algorithm. Assuming the β messages and the weight potentials are randomly, independently ordered, and a constant b, then the total running time for each iteration of belief propagation for b-matching with sufficient selection is $\mathcal{O}(N^{1.5})$, and the total running time to solve b-matching is $\mathcal{O}(N^{2.5})$.

It is important to point out the differences between the assumptions in Theorem 4.2.4 and why they do not always hold in real data scenarios. When nodes represent actual objects or entities and the weights are determined by a function between nodes, the weight values have dependencies and are therefore not completely randomly ordered. Furthermore, the β values change during belief propagation according to rules that depend on the weights, and in some cases can cause the selection time to grow to $\mathcal{O}(N)$. Nevertheless, in many sampling settings and real data generating processes, the weights are random enough and the messages behave well enough that sufficient selection yields significant speed improvements.

Finally, the space requirement for this algorithm has been reduced from the $\mathcal{O}(N^2)$ beliefs (or messages) of the previous belief propagation algorithm to $\mathcal{O}(N)$ storage for the α and β val-

ues of each row. Naturally, this improvement is most beneficial in settings where the weights are computable from an efficient function, whereas if the weights are arbitrary, they must be explicitly stored at the cost of $\mathcal{O}(N^2)$ memory. In most machine learning applications, however, the weights are computed from functions of node descriptor pairs, such as Euclidean distance between vectors or kernel values. In these applications, the algorithm needs only to store the node descriptors, the α and β values and, during the computation of Algorithm 3, $\mathcal{O}(N)$ beliefs (which can be immediately deleted before computing the next row). The weight cache adds $\mathcal{O}(cN)$ space, where we consider c a user-selected constant. Appendix A includes some additional heuristics for implementation of the belief propagation b-matching algorithm.

The space reduction is also significant for the purposes of parallelization. The computation of belief propagation is easy to parallelize, but the communication costs between processors can be prohibitive. With the proposed algorithm, each computer in a cluster stores only a copy of the node descriptors and the current α and β values. At each iteration, the cluster must share the $2N$ updated α and β values. This is in contrast to previous formulations where $\mathcal{O}(N^2)$ messages or beliefs needed to be transmitted between computers at each iteration for full parallelization. Thus, when it is possible to provide each computer with a copy of the node descriptor data, an easy parallelization scheme is to split the row updates between cluster computers at each iteration.

4.2.4 Parallel computation

At each iteration, each node independently updates its beliefs by running Algorithm 3 given the previous iterations' α and β vectors. This process is easily parallelizable by delegating the selection operations for nodes to different processors. Using this simple parallelization scheme, each computer stores the weights for the nodes it is responsible for (or the node descriptors), and a central computer collects and distributes the computed α and β vectors. At each iteration, the central computer sends the latest belief vectors to each worker, as well as the current b-matching assignment for the nodes each worker is responsible for. After each worker computes its new belief values, it sends these new belief values and the new b-matching assignments back to the central computer. The communication cost is significantly reduced by the unrolled recursion, resulting in the central computer sending an $\mathcal{O}(N)$ vector to each node at each iteration and receiving $\mathcal{O}(1)$ data back after computation. The pseudocode for such a parallelization scheme is nearly identical to Algorithm 2,

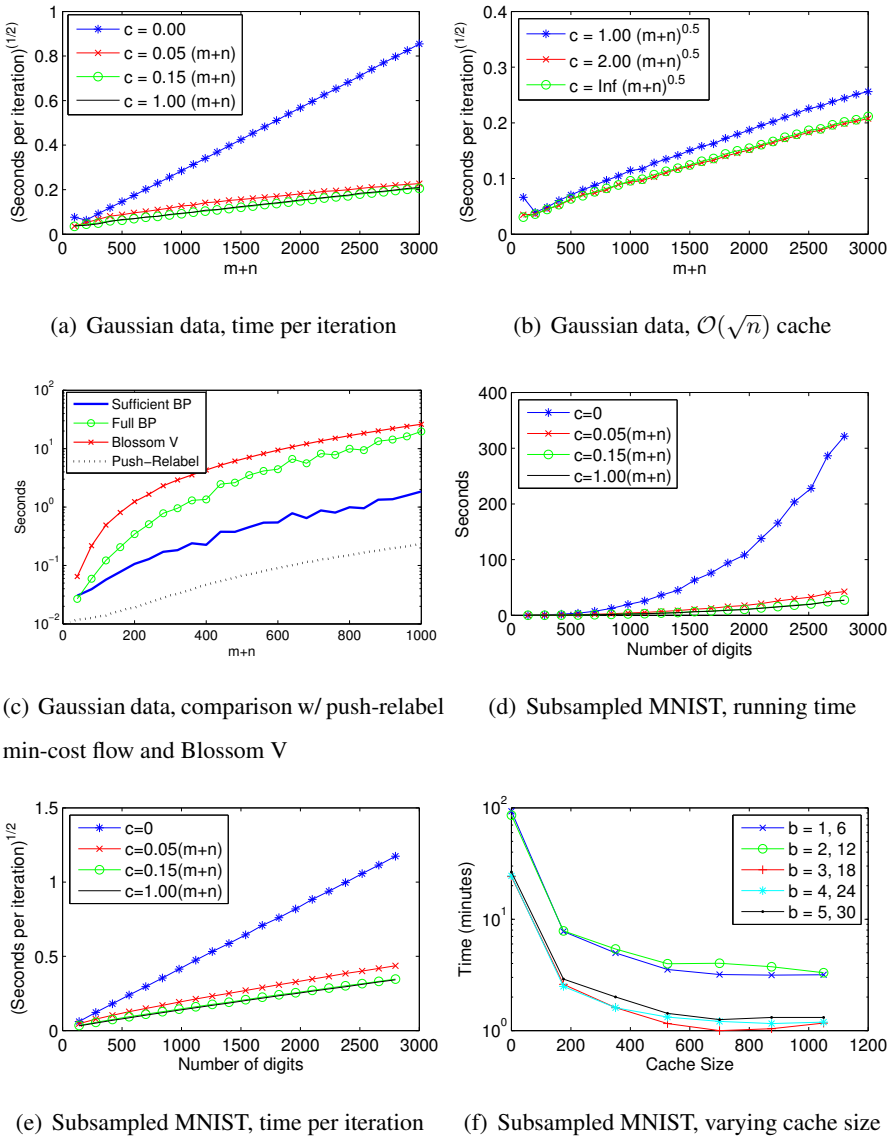
except the for-loop distributes work to different nodes rather than iterating sequentially.

To avoid centralizing the process, each computer can store all of the node descriptors (or only the weights associated with assigned nodes) as well as belief vectors, sending and receiving updates from all other nodes at each iteration. In this case, each node sends $\mathcal{O}(1)$ data to each other node at each iteration, which results in an overall bandwidth usage of $\mathcal{O}(N^2)$ per iteration, the same as in the centralized version. A decentralized version makes pre-sorting the β vector for sufficient selection more difficult, and including a decentralized parallel sort procedure that is efficient in practical parallel computing scenarios remains future work. Note that in cases where either parallel variant cannot perform exact synchronization, the belief propagation algorithm is still guaranteed to converge [Bayati *et al.*, To appear].

4.2.5 Empirical evaluation of sufficient-selection belief propagation for b-matching

Running time: synthetic and real data This section contains the results from experiments exploring the scaling behavior of the belief propagation algorithm for perfect bipartite b-matching. In particular, the experiments will show that the theoretical expected running time accurately predicts a significant running time improvement in practice on both synthetic weights and weights based on real data. The running time of the sufficient selection algorithm is measured and compared against three baseline methods: the standard belief propagation algorithm, which is equivalent to setting the proposed algorithm's cache size to zero, a compact C implementation of a push-relabel min-cost flow algorithm for the assignment problem [Goldberg and Kennedy, 1995], and, for reference, the Blossom V code by Kolmogorov [Kolmogorov, 2009], which is considered to be a state-of-the-art maximum weight non-bipartite matching solver. The non-bipartite matching problem is a significantly harder combinatorial optimization than the bipartite matching problem, but Blossom V is extremely optimized and provides another reference point for comparison.

For all experiments, node descriptors are sampled from zero-mean, spherical Gaussian distributions with variance 1.0, the weight function returns negative Euclidean distance, and we sample bipartitions of equal size ($m = n = N/2$). In the first experiment, points are sampled from $\mathbb{R}^{20}$. Using different cache sizes, the running time of the algorithm is measured for varying point set sizes from 10 to 3000. We set $b_i = 1, \forall i$. We measure the running time using actual CPU time. The square roots of per-iteration running times are drawn in Figure 4.3(a). For a cache size of zero,



(a) Gaussian data, time per iteration

(b) Gaussian data, $\mathcal{O}(\sqrt{n})$ cache

(c) Gaussian data, comparison w/ push-relabel min-cost flow and Blossom V

(d) Subsampled MNIST, running time

(e) Subsampled MNIST, time per iteration

(f) Subsampled MNIST, varying cache size

Figure 4.3: Timing results of fast belief propagation on various input formats. Figures 4.3(a) and 4.3(b) compare different cache sizes, where in 4.3(a), the cache size a constant scaling of the input size, and in 4.3(b), the cache size is a scaling of the square root of the input size. Figure 4.3(c) compares 1-matchings against a min-cost flow solver and the Blossom V algorithm. Figure 4.3(d) shows the solution time for subsampled MNIST digits of different sizes, and Figure 4.3(e) plots the root time per iteration as in the synthetic plots. Finally, for 10% of the full MNIST set, Figure 4.3(f) plots the total solution time for varying cache sizes. Diminishing returns are evident for higher cache sizes. See Table 4.1 for results on the full MNIST set.

where the algorithm is default belief propagation, the running time per iteration scales quadratically and that for non-zero cache sizes, the running time scales sub-quadratically. This implies that, at least for random, *iid*, Gaussian data and Euclidean weights, the weights and β values are uncorrelated enough to achieve the random permutation case speedup.

For the second experiment, node descriptors are drawn from $\mathbb{R}^5$, and we compare 1-matching performance between sufficient selection belief propagation, full belief propagation, push-relabel flow, and the Blossom V code by Kolmogorov. For sufficient selection, we set the cache size to $c = 2\sqrt{m+n}$, varying the point set size from 10 to 1000. In this case, there is no equivalent notion of per-iteration time for the combinatorial solvers, so we compare the full solution time. The min-cost flow solver is the fastest by a significant factor, with interactive-speed solution times at these graph-sizes, and the sufficient selection method brings the belief propagation algorithm much closer to the minimal scaling behavior of the min-cost flow solver.

All running time tests on synthetic data were run on a personal computer with an 8-core 3 GHz Intel Xeon processor (though each run was single-threaded). Figure 4.3 contains plots of all the running time experiments discussed in this section.

Handwritten digits We perform timing tests on the MNIST digits data set [LeCun *et al.*, 2001], which contains 60k training and 10k testing handwritten digit images. The images are centered, and represented as 28×28 pixel grayscale images. We use principle components analysis (PCA) to reduce the 784 pixel dimensions of each image to the top 100 principle eigenvector projections. We use negative Euclidean distance between PCA-projected digits as edge weights, and time sufficient selection belief propagation on a subsampled data set with varying cache sizes. In particular, for this test, we sample 10% of both the training and testing sets, resulting in 6000 training and 1000 testing digits. We generate feasible b-matching constraints by setting the target degree $b_{\mathrm{tr}} \in \{1, \ldots, 5\}$ for the training points and the target degree b_{te} for testing points to $b_{\mathrm{te}} = 6b_{\mathrm{tr}}$ (since there are six times as many training points).

Since there are a large number of candidate edges between training and testing examples, any algorithm that stores and updates beliefs or states for each edge, such as the original belief propagation algorithm [Huang and Jebara, 2007] or the Blossom V algorithm [Kolmogorov, 2009] cannot be run on most computers without the use of expensive virtual memory swapping. Thus, we only

compare the running times of linear memory b-matching belief propagation as described in Section 4.2.2 using different cache sizes. Using random subsamples of the MNIST data set, the average running times are plotted in Figure 4.3 against different input sizes and different cache sizes.

These timing tests were run on a Mac Pro with an 8-core 3 GHz Intel Xeon processor, each b-matching job running on only a single core. The results show that for a cache size of 200, the solution time is reduced from around an hour to fewer than ten minutes. Interestingly, the running time for larger b values is less, which is because belief propagation seems to converge in fewer iterations. For larger cache sizes, we achieve minimal further improvement in running time; it seems that once the cache size is large enough, the algorithm finishes selection before running out of cached weights.

Finally, using a cache size of 3500, finding the minimum distance matching for the full MNIST data set, which contains six hundred million candidate edges between training and testing examples, took approximately five hours for $b_{\mathrm{tr}} = 1$ and $b_{\mathrm{tr}} = 4$. The statistics from each run are summarized in Table 4.1. As in the synthetic examples, we count the number of belief lookups during the entire run and can compare against the total number that would have been necessary had a standard selection algorithm been used (which is $(m+n)^2$ per iteration). The running time is approximately 100 times faster than the estimated time for belief propagation with naive selection.

Table 4.1: Running time statistics on the full MNIST data set. Matching the full MNIST training set to the testing set considers 7000 nodes and 600 million edges. The table columns are, from left to right, the target degrees b_{tr} and b_{te} for training and testing nodes, raw running time for b-matching in minutes, the total number of belief lookups during the entire run, and the percentage of the belief lookups that would have been necessary using naive belief propagation ($\%$ Full).

b_{tr}	b_{te}	Time (min.)	Belief Lookups	$\%$ Full
1	6	285.77	4.5992×10^{10}	0.94%
4	24	306.76	5.2208×10^{10}	1.11%

4.3 Summary of fast belief propagation for maximum weight b-matching

This chapter details a belief propagation solver for maximum weight b-matching, including its derivation, theoretical analysis, and empirical evaluation. The algorithm is derived by constructing a Markov random field that encodes the objective function of maximum weight b-matching, then simplifying the max-product algorithm by exploiting redundancy and structure in the update formulas. The resulting algorithm is lightweight, parallelizable, and theoretically interesting for its guaranteed convergence.

In the context of this thesis, this solver serves as a driver for degree-based subgraph estimation, combining the procedure in Chapter 3 with the algorithm in this section. The next part of the thesis is a series of chapters describing various applications using degree-based subgraph estimation with a belief propagation driver. Additionally, Appendix A contains various heuristics to further improve the scalability of the belief propagation b-matching solver.

Part II

Applications

Chapter 5

Graph-Based Machine Learning

In addition to direct application for network analysis, graph-based methods are effective for classical learning and information retrieval applications. Maximum weight b-matching is useful for manifold learning, graph embedding, clustering, semi-supervised learning, and content and advertisement distribution [De Francisci Morales *et al.*, 2011; Jebara and Shchogolev, 2006; Jebara *et al.*, 2009; Mehta *et al.*, 2007; Shaw and Jebara, 2007; Shaw and Jebara, 2009]. For the problem of *classification*, in which the task is to label data points, an effective graph-based strategy is to construct a graph connecting similar points and *propagate* labels along the edges, from labeled points to unlabeled points. In the *supervised learning* setting, a set of labeled training points is used to predict labels for a set of unlabeled testing points. In the *semi-supervised learning* setting, a third set of unlabeled points is available for which predictions are not needed, but may be used to better characterize the distribution over data.

For various reasons, the common approach for graph construction is k-nearest neighbors. The natural intuition behind this approach is that each unlabeled point is likely to share the same label with its k nearest neighbors. This intuition primarily comes from human experience, where distances are well behaved in the natural two-dimensional or three-dimensional world. However, various factors such as high-dimensional geometry cause these intuitions to misrepresent behavior in real data. This chapter provides experimental evidence that other graph construction algorithms in the family of degree-based matching have various advantages over k-nearest neighbors.

5.1 Robustness of k-nearest neighbors vs. b-matching in high dimensional data

One common application of graph structure estimation is classification, or prediction of labels given data, where graph edges indicate label equality (or label equality votes). The common approach for classification of this form is to find the k nearest neighbors of each testing point and transductively predict the most popular label among its neighbors. Typically, k-nearest neighbors is the only degree requirement used, while various other options are available, especially in light of our contribution. The number of neighbors k is generally selected via cross-validation, but other priors may yield better performance. In this section, we provide some empirical evidence that the choice of degree prior, specifically b-matching versus k-nearest neighbors, can strongly influence classification accuracy.

Effects of high-dimensionality Since k-nearest neighbors greedily connects testing points to their nearest neighbors, undesirable behavior can occur in high dimensional data. Loosely, high-dimensionality allows more points to consider any given point their nearest neighbor, which means the label of any training point can affect more testing predictions with higher dimensionality. Figure 5.1 illustrates this behavior in two-dimensional space.

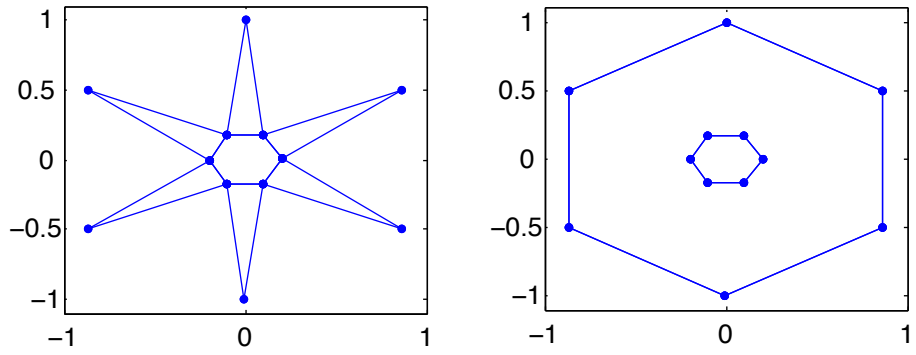
A generalization bound by Devroye and Wagner [Devroye and Wagner, 1979] provides some more insight for this behavior.

Theorem 5.1.1 ([Devroye and Wagner, 1979])**.** *Let the L_n denote the true risk (probability of misclassification), and let L_n^D denote the leave-one-out estimate of the risk. Then for n points in d dimensional space, the deviation between the estimated risk and the true risk is bounded by*

$$\Pr(|L_n^D - L_n| \geq \epsilon) \leq 2\exp(-n\epsilon^2/18) + 6\exp(-n\epsilon^3/108k(2+\gamma_d)),$$

where γ_d is the kissing number*, or the number of non-intersecting hyperspheres of equal radius that can touch a single hypersphere in d dimensions.*

The kissing number γ_d is known to grow exponentially with dimensionality [Conway *et al.*, 1987], so as dimensionality grows, the amount of data necessary for good generalization also grows exponentially. In the case of nearest neighbor classification, the kissing number corresponds to the total number of points that may consider the centroid of the central hypersphere to be their nearest



(a) k-nearest neighbors (b) b-matching

Figure 5.1: Illustration of greedy k-nearest neighbors against b-matching. Points on the outer ring are closer in Euclidean distance to points on the inner ring than other outer points. Using a target degree of 2, k-nearest neighbors connect four points to each inner point. Connecting points with b-matching ($b = 2$) preserves inner and outer ring membership across neighbors since the inner points cannot have more than two neighbors each.

neighbor. Thus, the generalization argument stems from the inherent instability of nearest-neighbor classification as dimensionality grows.

One strategy to counter-act the effects of high-dimensionality in k-nearest neighbor classification is to explicitly limit the number of testing points that may consider any given training point its graph neighbor to a constant b, which can be posed as a minimum distance b-matching. This limits the effects of any label change, since each training point can only vote on at most b testing predictions.

Empirical evaluation The DOROTHEA data set [Asuncion and Newman, 2007] is a database of chemical compounds represented by structural molecular features, which are to be classified as active (binding to thrombin) or inactive. The training set contains 800 examples and the validation set contains 350 examples. Each example is described by a tremendous 100,000 features, some of which were synthetically generated noise [8].

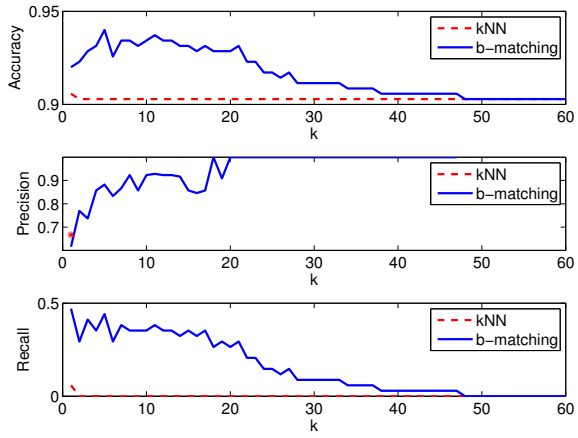
[8]The data set was used in 2001's KDD Cup Knowledge Discovery in Data Mining challenge, where the original competition goal was feature-selection.

Comparing k-nearest neighbor classification, where each testing point chooses its k neighbors by Euclidean distance and predicts the most popular label among its neighbors, against a form of b-matching, where each testing point must have b neighbors, and each training point must have no more than b neighbors, Figure 5.2 plots for various k and b values the classification accuracy, the precision, and the recall of positive examples. The figure also contains plots of the degree of the most popular training node as k and b grow. The difference in the maximum degree is significant, and it is caused by the high dimensionality of the data. We hypothesize that the b-matching performance is better because noisy points are limited in how many testing labels they can influence. In other words, with k-nearest neighbors, a training point may vote on all testing points, while with b-matching, training points cannot connect to more than b neighbors. Thus, if a training point is an outlier, its noisy influence using b-matching is more limited.

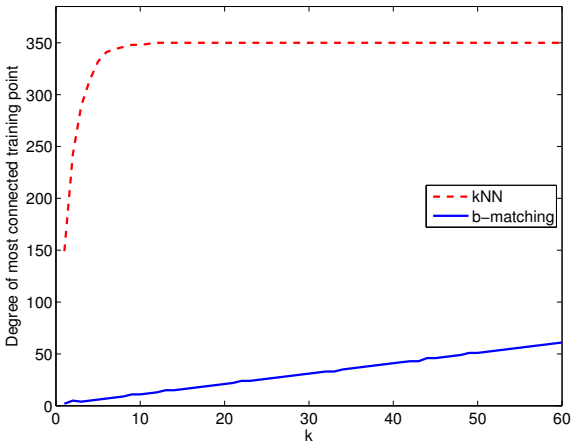
5.2 Robustness of b-matching against concept drift via translation

The stability of b-matchings also improves classification performance in cases where testing data has been translated due to *concept drift* [Huang and Jebara, 2007]. This section describes a synthetic experiment where concept drift cripples k-nearest neighbors, but b-matching can classify at near-perfect accuracy. An additional experiment on the more realistic setting of digit classification when the background of the digit changes between training and testing data, which is similar to translation in the vector space representation of digit images, shows the translation robustness of b-matching to a realistic computer vision problem.

Synthetic data We create synthetic data by sampling 50 training data points from two spherical Gaussians with means at $(3, 3)$ and $(-3, -3)$. We then sample 50 testing data points from similar Gaussians, but translated along the x-axis by $+8$. This is a severe case of translation where it is obvious that k-nearest neighbors will be fooled by the proximity of the testing Gaussian to the training Gaussian of the wrong class. Figure 5.3 shows the points we sampled from this setup. As expected, k-nearest neighbors only achieves 65% accuracy on this data for the optimal setting of k, while b-matching classifies the points perfectly for all settings of b.



(a) Accuracy metrics



(b) Maximum out-degree

Figure 5.2: Results comparing k-nearest neighbors to b-matching for classification on the DOROTHEA data set. Top: accuracy, precision and recall for varying values of k. The b-matching prior outperforms k-nearest neighbors on all metrics for nearly all settings of b and k. The precision for k-nearest neighbors becomes undefined for k greater than 1 because all predicted labels are negative. Bottom: the maximum number of neighbors of a training point. This plot clearly shows that k-nearest neighbors is connecting certain training points to tremendous amounts of testing points, skewing the voting and allowing significant noise if any of these hub points are mislabeled or outliers.

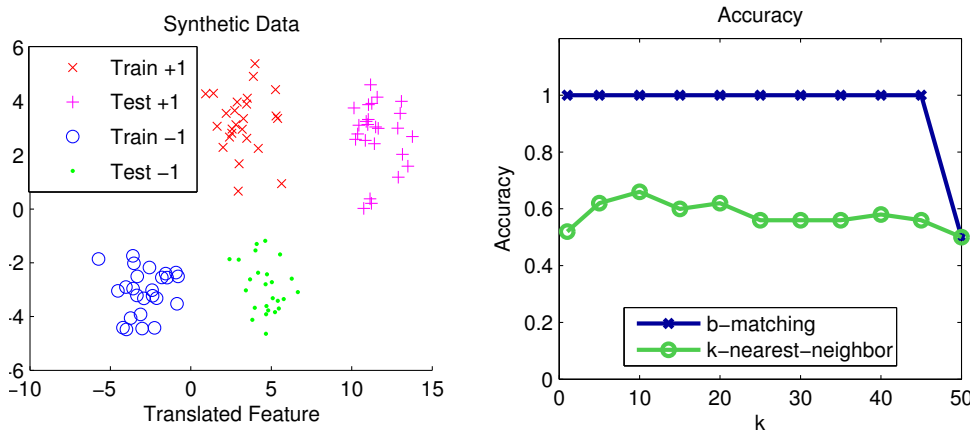


Figure 5.3: Left: Synthetic classification data where the test data is translated to the right along the x-axis. Right: Accuracy for k-nearest neighbor versus b-matching for various k (or b).

Digit backgrounds For a more realistic example of this phenomenon, we sample the MNIST digits dataset of 28×28 grayscale digits. For each of the similar-looking digits 3, 5, and 8, we sample 100 training points from the training set and 100 from the testing set. We average accuracy over 20 random samplings to avoid anomalous results. We cross-validate over settings of b and k in the range $[1, 300]$ and save the best accuracy achieved for each algorithm on each sampling.

We examine the case where training and testing data are collected under drastically different conditions by simulating that the testing digits are printed against various backgrounds. We replace all white (background) pixels in the testing images with these backgrounds and attempt classification using both algorithms on these new datasets. Figure 5.4 contains examples of digits placed in front of backgrounds.

The results show that b-matching outperforms k-nearest neighbors, sometimes by significant margins. On the unaltered white background and the diagonal lines, both algorithms classify at slightly higher than 90% accuracy. On the grid, white noise, brushed metal, wood and marble backgrounds, b-matching is invariant to the transformation while k-nearest neighbors suffers drops in accuracy. Figure 5.4 contains a plot of both algorithms' average accuracies for each background type.

Since the background replacement is like a translation of the image vectors that preserves the general shape of the distribution, b-matching is a more useful tool for connecting training examples

to testing points than k-nearest neighbors.

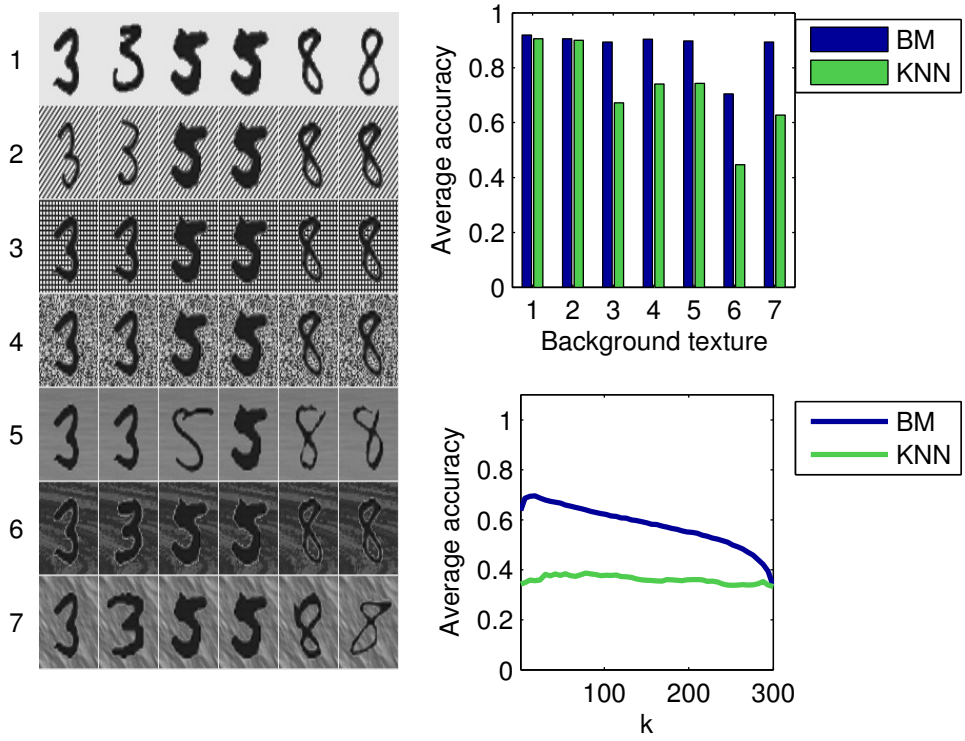


Figure 5.4: (Left) Examples of digits placed on backgrounds: 1-unaltered, 2-diagonal lines, 3-grid, 4-white noise, 5-brushed metal, 6-wood, 7-marble. (Top right) Average accuracies over 20 random samplings for optimal setting of b or k. (Bottom right) Average accuracy on wood background for various settings of b or k.

5.3 Graph construction for semi-supervised learning

This section reproduces a subset of experiments by Jebara *et al*. comparing b-matching to k-nearest neighbors in the task of graph-based semi-supervised learning [Jebara *et al.*, 2009]. In semi-supervised learning, a prediction algorithm is given labeled training data, unlabeled background data, and unlabeled query data to be labeled by the predictor. Various approaches to solve this problem construct a graph, connecting all points — labeled, unlabeled, and query points — based on similarity, and perform some form of *label propagation*. Analogous to voting in supervised

k-nearest neighbor classification, label propagation algorithms transfer information from labeled points to all unlabeled points, where in the semi-supervised setting, the unlabeled background points are not tested, but instead help characterize the space of all points.

As in the original experiments, we use the USPS digit data set, which contains images of single digits labeled 0 through 9. Sampling a random subset of 2000 digits, we randomly select a single labeled instance per class. We use either k-nearest neighbors or approximate non-bipartite b-matching via belief propagation to construct a minimum Euclidean distance graph between digits, with $k = b = 7$. After constructing the graph, we weight each edge according to a *Gaussian kernel re-weighting*, in which the edge weight between feature vectors x_i and x_j is

$$W_{ij} = \exp\left(\frac{d(\mathbf{x}_i, \mathbf{x}_j)}{2\sigma^2}\right),$$

where in these experiments we choose σ to be $\bar{d}_k/3$, where $\bar{d}_k$ is the average distance between each point and its k'th nearest neighbor [Jebara *et al.*, 2009].

Given the constructed graph, three label propagation methods are compared: *Gaussian random fields* (GRF) [Zhu *et al.*, 2003], *local and global consistency* (LGC) [Zhou *et al.*, 2004], and *graph transduction via alternating minimization* (GTAM) [Wang *et al.*, 2008]. The average error rates for each label propagation method and each graph construction method over 20 random splits are listed in Table 5.1, where b-matching produces a lower error rate than k-nearest neighbors for all algorithms. Similar results on more thorough experiments are presented by Jebara *et al.* [Jebara *et al.*, 2009].

	GRF	LGC	GTAM
k-nearest neighbors	0.175550	0.146625	0.126000
b-matching	0.167625	0.132925	0.115575

Table 5.1: Error rates of USPS semi-supervised classification.

5.4 Summary of graph-based classification

This section provides evidence that b-matching is a useful graph-construction option for graph-based learning. The benefits of b-matching are demonstrated for classification problems in high-

dimensionality, when concept-drift causes training and testing data to differ, and in the semi-supervised setting.

Chapter 6

Collaborative Filtering

At the time of this thesis work, in 2011, the Internet is booming with countless sources of information, constantly producing more content as well as more tools for the acquisition or delivery of old and new content to users. Growth in these areas has caused a severe need for filtering of the available content to avoid overwhelming users. *Collaborative filtering* is the general problem of filtering content using user interaction data. The collaborative filtering problem has been gradually formalized in recent years. The Netflix Grand Challenge [Bell and Koren, 2007; Koren *et al.*, 2009] widely publicized the problem to mainstream media as well as research communities around machine learning.

The collaborative filtering task typically concerns the prediction of ratings of users for items, using information such as past ratings by the users for other items or by other users for the queried items, or features of the users and items in question. For example, on the Netflix service, users rate movies, television shows, and videos, and part of the Netflix experience is recommendations of movies for users to watch next.

This chapter addresses the collaborative filtering problem as a relational learning problem. Measured interactions between users and items can be viewed as edges or relations, and the goal of predicting relations between unseen user-item pairs is posed as a graph estimation. Initially, Section 6.2 handles a simplified form of the collaborative filtering problem in which user interactions are measured as binary, positive or negative, ratings. Section 6.3 extends the ideas from 6.2 to the more general, multi-relational setting, in which multi-valued ratings are modeled as different relations between users and items, in a framework which also allows continuous-valued relations.

6.1 Prior approaches for collaborative filtering

Early approaches for collaborative filtering involved natural ideas such as clustering [Breese *et al.*, 1998; Ungar and Foster, 1998] and simple graphical models [Hofmann and Puzicha, 1999; Marlin, 2004]. More recently, a popular approach is to model the observable ratings by users for items as entries in a partially-observed *rating matrix*. The idea for predicting the remaining, unobserved entries in the matrix is to assume that the rating matrix is a matrix product of latent user and item factor matrices. The basic learning algorithm is then to factorize the rating matrix into these factors, using variants of *singular value decomposition* (SVD). To regularize the predictions of these methods, the factor matrices are assumed to be low-dimensional, thus limiting the complexity of the learned model. Various important empirical and theoretical results not discussed in this thesis have resulted from variations of this basic idea[Bell and Koren, 2007; Breese *et al.*, 1998; Koren *et al.*, 2009; Lim and Teh, 2007; Montanari *et al.*, 2009; Rennie and Srebro, 2005; Salakhutdinov and Mnih, 2008b; Salakhutdinov and Mnih, 2008a; Srebro *et al.*, 2004; Weimer *et al.*, 2007].

6.2 Collaborative filtering as graph estimation

We apply degree-based subgraph estimation as a post-processing step in a graph prediction problem. Consider the task of predicting a graph defined by the recommendations of items by users in a variation of the standard collaborative filtering setting. We define a *recommendation graph* as a bipartite graph between a set of users $U = \{u_1, \ldots, u_m\}$ and a set of items $V = \{v_1, \ldots, v_n\}$ that the users have rated with binary recommendations. We assume a sparse rating matrix $\mathbf{X} = \{0, 1\}^{m \times n}$ representing the recommendations of items by users such that entry X_{ij} is 1 if the i'th user recommends the j'th item and is otherwise 0. The rating matrix $\mathbf{X}$ is equivalent to the adjacency matrix of the recommendation graph. The training data is a set T of user-item pairs and whether an edge is present between their nodes in the recommendation graph. The testing data is another set Q of *queried* user-item pairs, and the task is to predict which of the testing pairs will have a preference edge present.

First, we provide motivation for using degree priors in post-processing. The degrees of nodes in the predicted graph represent the number of items liked by a user or the number of users that

like an item. Under certain assumptions, the rate of liking or being liked will concentrate around its empirical estimate, and the deviation probability between training and testing rates is bounded by a log-concave upper bound. Therefore, we can use the deviation bound as a degree prior to post-process predictions output by a state-of-the-art inference method. This, in effect, forces predictions to obey the bounds.

6.2.1 Concentration bound

We assume that users U and items V are drawn *iid* from arbitrary population distributions $\mathbb{D}_u$ and $\mathbb{D}_v$. We also assume that the probability of an edge between any nodes u_i and v_j is determined by a function that maps the features of the nodes to a valid Bernoulli probability

$$\Pr(X_{ij} = 1|u_i, v_j) = g(u_i, v_j) \in [0, 1]. \tag{6.1}$$

These assumptions yield a natural dependency structure for rating probabilities. The joint probability of users, items and ratings is defined as follows:

$$\Pr(\mathbf{X}, U, V|\mathbb{D}_u, \mathbb{D}_v) \propto \prod_{ij} \Pr(X_{ij}|u_i, v_j) \prod_i \Pr(u_i|\mathbb{D}_u) \prod_j \Pr(v_j|\mathbb{D}_v). \tag{6.2}$$

The structure of this generative model implies dependencies between the unobserved ratings and even dependencies between the users and items. This is because the query rating variables and all user and item variables are latent. Due to the independent sampling procedure on users and items, this is known as a hierarchical model [Gelman *et al.*, 2003] and induces a coupling, or interdependence, among the test predictions that are to be estimated by the algorithm. Since the rating variables exist in a lattice of common parents, this dependency structure and the hierarchical model are difficult to handle in a Bayesian setting unless strong parametric assumptions are imposed. Instead, we next derive a bound that captures the interdependence of the structured output variables $\mathbf{X}$ without parametric assumptions.

We assume that both the training and testing user-item sets are completely randomly revealed from a set of volunteered ratings, which allows proof of an upper bound for the probability that the empirical edge rate of a particular node deviates between training and testing data. In other words, we estimate the probability that an empirical row or column average in the adjacency matrix deviates from its true mean. Let $X_i = \{X_{ij}|ij \in T \cup Q\}$ represent the row of training and query ratings by

user i. Let function $\Delta(X_i)$ represent the difference between the training and query averages,

$$\Delta(X_i) = \frac{1}{m_i} \sum_{j|ij \in T} X_{ij} - \frac{1}{\hat{m}_i} \sum_{j|ij \in Q} X_{ij},$$

which is bounded by the following theorem.

Theorem 6.2.1. *Given that users $U = \{u_1, \ldots, u_m\}$ and rated items $V = \{v_1, \ldots, v_n\}$ are drawn* iid *from arbitrary distributions $\mathbb{D}_u$ and $\mathbb{D}_v$ and that the probability of positive rating by a user for an item is determined by a function $g(u_i, v_j) \mapsto [0, 1]$, the average of query ratings by each user is concentrated around the average of his or her training ratings. Formally,*

$$\begin{aligned} \Pr\left(\Delta(X_i) \geq \epsilon\right) &\leq 2\exp\left(-\frac{\epsilon^2 m_i \hat{m}_i}{2(m_i + \hat{m}_i)}\right), \\ \Pr\left(\Delta(X_i) \leq -\epsilon\right) &\leq 2\exp\left(-\frac{\epsilon^2 m_i \hat{m}_i}{2(m_i + \hat{m}_i)}\right). \end{aligned} \tag{6.3}$$

The proof of Theorem 6.2.1 is available in appendix Section B.3, as well as the equivalent corollary bounding the deviation of ratings per item.

Many collaborative filtering learning algorithms applied in this binary setting estimate the edge likelihoods. However, independently predicting the most likely setting of each edge is equivalent to using a uniform prior over the rating averages. A uniform prior violates the bound at a large enough deviation from the training averages. Specifically, this occurs for users or items with a large number of training and testing examples. Thus, it may be advantageous to use a prior that obeys the bound. Since the bound decays quadratically in the exponent, priors that will never violate the bound must decay at a faster rate. These exclude uniform and Laplace distributions and include Gaussian, sub-Gaussian and delta distributions. We propose simply using the normalized bound as a prior.

6.2.2 Edge weights

To learn reasonable values for the independent edge weights, we use *fast max-margin matrix factorization* (fMMMF) [Rennie and Srebro, 2005] using a logistic loss function, which has a natural probabilistic interpretation [Rennie, 2007]. In the binary-ratings setting, the gradient optimization for logistic fMMMF, which uses a logistic loss as a differentiable approximation of hinge-loss, can be interpreted as maximizing the conditional likelihood of a generative model that is very similar to

one discussed above. The objective is[9]

$$\min_{U,V} J(\mathbf{U},\mathbf{V}) \quad = \quad \frac{1}{2}(||\mathbf{U}||^2_{\text{Fro}} + ||\mathbf{V}||^2_{\text{Fro}}) + C\sum_{ij} \log\left(1 + e^{-X^{\pm}_{ij}(\mathbf{u}_i^\top \mathbf{v}_j - \theta_i)}\right).$$

The probability function for positive ratings is the logistic function,

$$\Pr(X_{ij}|\mathbf{u}_i,\mathbf{v}_j,\theta_i) = g(\mathbf{u}_i,\mathbf{v}_j) = \frac{1}{1+e^{-(\mathbf{u}_i^\top \mathbf{v}_j - \theta_i)}},$$

which yields the exact loss term above. Minimization of squared Frobenius norm corresponds to placing zero-mean, spherical Gaussian priors on the $\mathbf{u}_i$ and $\mathbf{v}_j$ vectors, $\Pr(\mathbf{u}_i) \propto \exp(-\frac{1}{C}||\mathbf{u}_i||^2)$ and $\Pr(\mathbf{v}_j) \propto \exp(-\frac{1}{C}||\mathbf{v}_j||^2)$. This yields the interpretation of fMMMF as MAP estimation [Rennie, 2007]:

$$\max_{\mathbf{U},\mathbf{V},\Theta} \prod_{ij} P(X_{ij}|\mathbf{u}_i,\mathbf{v}_j,\theta_i) \prod_i \Pr(\mathbf{u}_i) \prod_j \Pr(\mathbf{v}_j).$$

From the estimated $\mathbf{U}$ and $\mathbf{V}$ matrices from fMMMF, we use the logistic probabilities to set the singleton functions over edges (*i.e.*, edge weights). Specifically, the weight of an edge is the change in log-likelihood caused by switching the edge from inactive to active, $W_{ij} = \mathbf{u}_i^\top \mathbf{v}_j - \theta_i$.

6.2.3 Results

These experiments test five data sets. Four are standard collaborative filtering datasets, thresholded at reasonable levels. The last is trust/distrust data gathered from Epinions.com which represents whether users trust other users' opinions. The EachMovie data set contains 2,811,983 integer ratings by 72,916 users for 1,628 movies ranging from 1 to 6, which we threshold at 4 or greater to represent a positive rating. The portion of the Jester data set [Goldberg *et al.*, 2001] we used contains 1,810,455 ratings by 24,983 users for 100 jokes ranging from -10 to 10, which we threshold at 0 or greater. The MovieLens-Million data set contains 1,000,209 integer ratings by 6,040 users for 3,952 movies ranging from 1 to 5, which we threshold at 4 or greater. The Book Crossing data set [Ziegler *et al.*, 2005] contains 433,669 explicit integer ratings[10] by 77,805 users for 185,854 books

[9]Here $X^{\pm}_{ij}$ represents the signed $\{-1,+1\}$ representation of the binary rating, whereas previously, we use the $\{0,1\}$ representation. Additionally, the matrix factorization notation treats U and V as matrices where the rows are vector descriptors of users and items.

[10]The Book Crossing data set contains many more "implicit" recommendations, which occur when users purchase books but do not explicitly rate them. Presumably, these indicate positive opinions of the books; however, it is unclear what defines a negative implicit rating, so we only experiment on the explicit ratings.

ranging from 1 to 10, which we threshold at 7 or greater. Lastly, the Epinions data set [Massa and Avesani, 2005] contains 841,372 trust/distrust ratings by 84,601 users for 95,318 authors.

Each data set is split randomly three times into half training and half testing ratings. We randomly set aside $1/5$ of the training set for validation, and train logistic fMMMF on the remainder using a range of regularization parameters. The output of fMMMF serves as both the baseline as well as the weight matrix of our algorithm. We set the degree prior for each row/column to be proportional to the deviation bound from Theorem 6.2.1. Specifically, we use the following formula to set the degree potential ψ_i for the i'th user:

$$\psi_i^{\text{user}}(k) = -\lambda \frac{\left(\frac{1}{m_i}\sum_{j|ij\in T} X_{ij} - k/\hat{m}_i\right)^2 m_i \hat{m}_i}{2(m_i + \hat{m}_i)}. \tag{6.4}$$

The appropriate substitutions form the equivalent formula for the degree potentials of the j'th item. We introduce a regularization parameter λ that scales the potentials. When λ is zero, the degree prior becomes uniform and the MAP solution is to threshold the weight matrix at 0 (the default fMMMF predictions). At greater values, we move from a uniform degree prior (default rounding) toward strict b-matching, following the shape of the concentration bound at intermediary settings. We explore increasing values of λ starting at 0 until either the priors are too strong and overfit or until the value of λ is so great that we are solving a simple b-matching with degrees locked to an integer value instead of a distribution of integers. Increasing λ thereafter will not change the result. We validate at this stage by including the testing and held-out validation ratings in the query set of ratings, and consider the regularization parameter that produces the best performance on the validation set. The post-processing procedure is described in Algorithm 4.

The running time of the post-processing procedure is short compared to the time spent learning edge weights via fMMMF. This is due to the fast belief propagation matching code and the sparsity of the graphs. Each graph estimation takes a few minutes (no more than five), while the gradient fMMMF takes hours on these large-scale data sets.

We compare the zero-one error of prediction on the data. In particular, the key comparison is between the fMMMF output that performed best on cross-validation data and the MAP solution of the same output with additional degree priors. The results indicate that adding degree priors reduces testing error on all splits of five data sets. The error rates are represented graphically in Fig. 6.1 and numerically in Table 6.1. With higher λ values, the priors pull the prediction averages closer to

Algorithm 4 Post-processing procedure for collaborative filtering using degree concentration.

Require: Partially observed rating matrix $\mathbf{X}$, training index-pairs T, query index-pairs Q, regularization parameter λ, collaborative filtering algorithm CF, degree-based subgraph estimator DBSE

1: $\mathbf{P} \leftarrow \mathrm{CF}(\mathbf{X}, T)$

2: $W_{ij} \leftarrow \log \frac{P_{ij}}{1-P_{ij}} \ \forall (i,j) \in Q$

3: $\psi_i^{\mathrm{user}}(k) \leftarrow -\lambda \left(\frac{1}{m_i} \sum_{j|ij \in T} X_{ij} - k/\hat{m}_i \right)^2 m_i \hat{m}_i / \left(2(m_i + \hat{m}_i) \right), \quad \forall i, k$

4: $\psi_j^{\mathrm{item}}(k) \leftarrow -\lambda \left(\frac{1}{n_j} \sum_{i|ij \in T} X_{ij} - k/\hat{n}_j \right)^2 n_j \hat{n}_j / \left(2(n_j + \hat{n}_j) \right), \quad \forall j, k$

5: $\hat{E} \leftarrow \mathrm{DBSE}(\mathbf{W}, Q, \psi)$

6: $X_{ij} \leftarrow I((i,j) \in \hat{E}), \quad \forall (i,j) \in Q$

7: **return** $\mathbf{X}$

the training averages, which causes overfitting on all but the Epinions data set. Interestingly, even b-matching the Epinions data set improves the prediction accuracy over fMMMF. This suggests that the way users decide whether they trust other users is determined by a process that is strongly concentrated. These experiments provide evidence that enforcing degree distribution properties on the estimated graph consistently improves the performance of a state-of-the-art factorization approach.

Data set	fMMMF	Degree
EachMovie	0.3150 ± 0.0002	$\mathbf{0.2976 \pm 0.0001}$
Jester	0.2769 ± 0.0008	0.2744 ± 0.0021
MovieLens	0.2813 ± 0.0004	$\mathbf{0.2770 \pm 0.0005}$
BookCrossing	0.2704 ± 0.0016	0.2697 ± 0.0016
Epinions	0.1117 ± 0.0005	$\mathbf{0.0932 \pm 0.0003}$

Table 6.1: Average zero-one error rates and standard deviations of best MAP with degree priors and fMMMF chosen via cross-validation. Average taken over three random splits of the data sets into testing and training data. Degree priors improve accuracy on all data sets, but statistically significant improvements according to a two-sample t-test with a rejection level of 0.01 are bold.

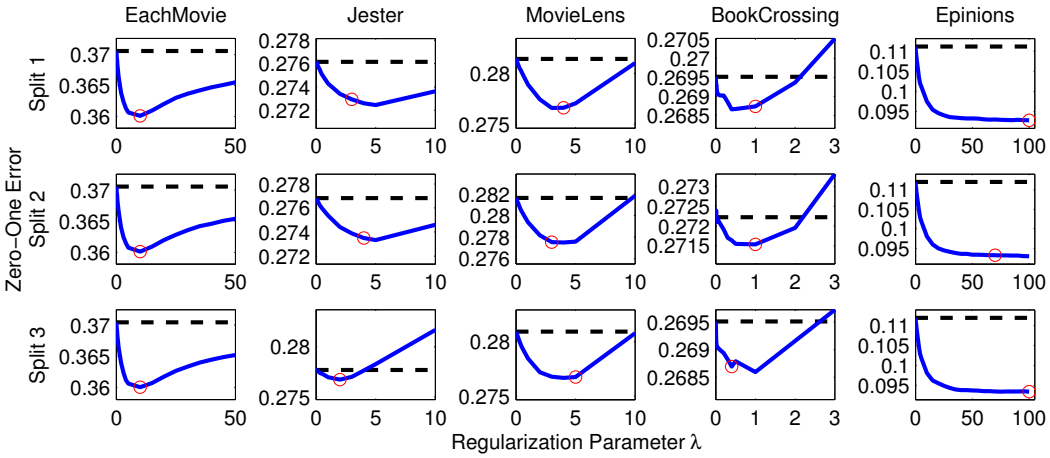


Figure 6.1: Collaborative filtering testing errors of MAP solution across different data sets and random splits. The horizontal axis of each plot represents the scaling parameter λ and the vertical axis represents the error rate. The solid blue line is the MAP solution with degree priors, and the dotted black line is the logistic-fMMMF baseline. The red circle marks the setting of λ that performed best on the cross-validation set. See Table 6.1 for the numerical scores.

6.3 Collaborative filtering via rating concentration

This section extends the ideas from the previous section to the more general collaborative filtering problem by again exploiting the concentration of user and item statistics. The statistics in this section may be relations between users and items, but the terminology in this section treats them as arbitrary functions, since the mathematical framework developed here allows modeling of any continuous bounded function, rather than the discrete, or binary relations typically used in relational learning.

By making a relatively agnostic assumption that users and items are drawn independently and identically-distributed (*iid*), it can be shown that the statistics of training ratings must concentrate close to the expectations of corresponding query ratings. Such assumptions are weaker than those used in previous approaches which often assume a parametric form on the generative model or assume that the rating matrix is low-rank. Nevertheless, an otherwise indifferent probability estimate that obeys such bounds (for instance the maximum entropy estimate) provides state-of-the-art performance [Huang and Jebara, 2010].

The method described here is largely complementary with current approaches since these leverage different intuitions, such as specific parametric forms for the distributions involved and low-rank constraints on the rating matrix. For instance, the assumption that the ratings matrix is low rank underlies many singular value decomposition (SVD) techniques. Therein, users and items are assumed to be *iid*, and ratings are assumed to be randomly revealed such that there is no bias between training and testing statistics. These assumptions have been shown to be unrealistic [Marlin *et al.*, 2007], however empirical performance remains promising. The SVD approach further assumes that the ratings are sampled from distributions parametrized by the inner products of low-dimensional descriptor vectors for each user and each item. In other words, the full rating matrix is assumed to be the product of a user population matrix and an item population matrix, possibly with additional independent noise. Often, such matrices are estimated with some form of regularization, either by truncating their rank, by penalizing their Frobenius norm or by placing Gaussian priors (a parametric assumption) on their descriptor vectors [Breese *et al.*, 1998; Lim and Teh, 2007; Montanari *et al.*, 2009; Rennie and Srebro, 2005; Salakhutdinov and Mnih, 2008a; Salakhutdinov and Mnih, 2008b; Srebro *et al.*, 2004; Weimer *et al.*, 2007].

Conversely, the approach in this section only makes minimal hierarchical sampling assump-

tions. It assumes that users and items are sampled *iid* and that each rating is subsequently sampled independently from a conditional probability distribution that depends on the respective user-item pair in an arbitrary manner. It is also assumed that the ratings are revealed randomly. The resulting learning algorithm makes no further assumptions about the distributions or the interaction between items and users (such as the inner product assumption most low-rank matrix-factorization methods make). Subsequently, we prove that, as long as each rating distribution depends only on the user and item involved, statistics from a user's (or item's) training data concentrate around the expected averages of the query probabilities. The result is a concentration inequality which holds regardless of modeling assumptions. The combination of these concentration inequalities defines a convex hull of allowable distributions. With high probability, the desired solution lives within this set but is otherwise underdetermined. A reasonable way to select a particular member of this set is to identify the one that achieves maximum entropy (or minimum relative entropy to a prior). The maximum entropy criterion is merely used as an agnostic regularizer to handle the underdetermined estimation problem and ensure the uniqueness of the recovered estimate within the convex hull.

Since the dependencies in the rating process exhibit a hierarchical structure, the method proposed is reminiscent of the hierarchical maximum entropy framework [Dudík *et al.*, 2007]. In fact, the proposed algorithm can be viewed as a specific application of hierarchical maximum entropy where we estimate distributions linked by common parents (from which we have no samples) using statistics gathered from separate distributions with one sample each. Thus, the collaborative filtering setting is an extreme case of the hierarchical maximum entropy setup since, without the hierarchy, there would be no information about certain components of the probability model. Moreover, previous work [Dudík *et al.*, 2007] proposed tree-structured hierarchies while this article explores a grid-structured (non-tree) hierarchy due to the matrix setup of users and items in the collaborative filtering problem.

The proposed intuitions and concentration inequalities of this section complement previous parametric approaches and provide additional structure to the collaborative filtering problem. They may be used in conjunction with other assumptions such as low-rank matrix constraints. Similarly, the concentration bounds hold whether the data is generated by a distribution with known parametric form or by any arbitrary distribution.

6.3.1 Algorithm description

Consider the collaborative filtering problem where the input is a partially observed rating matrix $\mathbf{X} \in \mathbb{Z}^{M \times N}$. Each matrix element $X_{ij} \in \{1, \ldots, K\}$ is a random variable representing the rating provided by the i'th user for the j'th item where $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$. The users $\{u_1, \ldots, u_m\}$ and the items $\{v_1, \ldots, v_n\}$ are variables drawn *iid* from arbitrary sample spaces $u_i \in \Omega_u$ and $v_j \in \Omega_v$, respectively. The observed ratings (where a sample of the random variable is provided) will be treated as a training set for the collaborative filtering problem and used to estimate unobserved ratings (where no sample of the random variable is available). The desired output is a set of predicted probability distributions on certain query ratings whose indices are specified a priori. Let T be the set of observed training (i, j) indices and let Q be the set of query indices. Given $\{X_{ij} | (i, j) \in T\}$ and Q, the goal is to estimate the probabilities $\{p(X_{ij} | u_i, v_j) | (i, j) \in Q\}$.

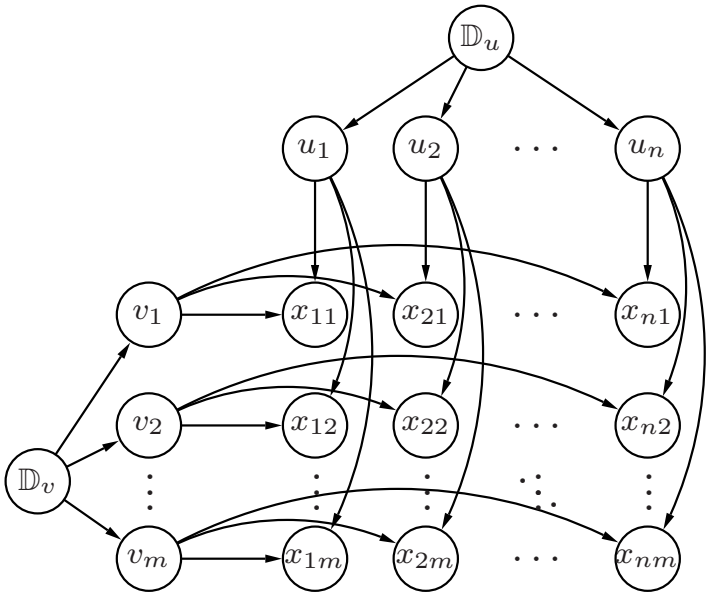


Figure 6.2: Graphical model of sampling assumptions. We solve for the probabilities of the query ratings without explicitly estimating the user and item descriptors.

Sampling assumptions A major challenge in the sampling setting of collaborative filtering is that only a single sample rating[11] is observed from the training distributions $\{p(X_{ij}|u_i, v_j)|(i,j) \in T\}$ and *zero* samples are observed from the query distributions $\{p(X_{ij}|u_i, v_j)|(i,j) \in Q\}$. To transfer information from training samples to the query distributions, it will be helpful to make a hierarchical sampling assumption. Figure 6.2 depicts a graphical model representation of the proposed structure that will be used. First, users are drawn *iid* from an unknown distribution $p(u)$ and items are drawn *iid* from another unknown distribution $p(v)$. Subsequently, for some pairs of users and items, a rating is drawn independently with dependence on the corresponding item and user samples (the rating's parents in the graphical model).

It is natural to require that the ratings are samples from multinomial distributions over a range of rating values. In most collaborative filtering data sets (*e.g.*, the Movielens data sets), ratings are discrete integer values (*e.g.*, 1 to 5), so the multinomial is non-restrictive. We further assume that the multinomial distributions are conditioned on a latent user descriptor variable $u_i \in \Omega_u$ and a latent item descriptor variable $v_j \in \Omega_v$ for each query $X_{ij} \in \{1, \ldots, K\}$. In other words, we assume rating samples are drawn from $p(X_{ij}|u_i, v_j) = g(X_{ij}, u_i, v_j)$, where g provides an arbitrary mapping of the user and item descriptor variables to a valid multinomial distribution in the probability simplex. This is in contrast to standard (SVD) assumptions, which require that the function is constrained to be $g(X_{ij}, u_i^\top v_j)$, where the function g may be constrained parametrically and must depend solely on the inner product of low-dimensional *vector* descriptors in a Euclidean space.

Ratings X_{ij} for different users and items in this formulation are not identically distributed, not even for any particular user or item. The distribution $p(X_{ij}|u_i, v_j)$ for user i for an item j can be dramatically different from $p(X_{ik}|u_i, v_k)$, user i's rating distribution for another item $k \neq j$. However, the sampling structure in Figure 6.2 allows the transfer of information across many distributions since users (and items) are sampled *iid* from a common distribution $p(u)$ (and $p(v)$). The joint probability distribution implied by the figure factorizes as $\prod_{ij} p(X_{ij}|u_i, v_j)p(u_i)p(v_j)$ and, in particular, we are interested in recovering $\prod_{(i,j)\in Q} p(X_{ij}|u_i, v_j)$.

The aforementioned sampling assumptions will establish that the empirical average of any function of the ratings generated by a single user (or item) is close to its expectation with high probabil-

[11] Recommendation data sets may include multiple ratings per user-item pair, though these are rare in practice.

ity. More specifically, empirical averages over training samples are close to corresponding averages over the expected values of the query distributions.

Concentration bound In this section, we present a theorem proving the concentration of training statistics to expected query averages. Specifically, we consider bounded scalar functions $f_k(X) \mapsto [0, 1]$ that take ratings as input and output a value inclusively between 0 and 1.[12] Examples of such functions include the normalized rating itself (*e.g.*, $(x-1)/(K-1)$, for ratings from 1 to K), or indicator functions for each possible value (*e.g.*, $I(x = 1)$).

We will consider bounding the difference of two quantities. The first quantity is the empirical average of function $f_k(X)$ over the training ratings. Since this quantity is fixed throughout learning, we simplify notation by using μ_{ik} to denote this average of function $f_k(X)$ for user i's ratings, and using ν_{jk} to denote the average for item j's ratings. Let m_i be the number of training ratings for user i and let n_j be the number of training ratings for item j. These averages are then

$$\mu_{ik} = \frac{1}{m_i} \sum_{j|(i,j)\in T} f_k(X_{ij}), \qquad \nu_{jk} = \frac{1}{n_j} \sum_{i|(i,j)\in T} f_k(X_{ij}). \tag{6.5}$$

The second quantity of interest is the expected average of $f_k(X)$ evaluated on the query ratings. Let $\hat{m}_i$ be the number of query ratings for user i and $\hat{n}_j$ be the number of query ratings for item j. The expected averages are then expressed as

$$\frac{1}{\hat{m}_i} \sum_{j|(i,j)\in Q} \mathbb{E}_{p(X_{ij}|u_i,v_j)}[f_k(X_{ij})], \qquad \frac{1}{\hat{n}_j} \sum_{i|(i,j)\in Q} \mathbb{E}_{p(X_{ij}|u_i,v_j)}[f_k(X_{ij})]. \tag{6.6}$$

The following theorem bounds the differences between the quantities in Equation (6.5) and Equation (6.6).

Theorem 6.3.1. *For the ratings of user i, the difference*

$$\epsilon_{ik} \quad = \quad \frac{1}{m_i} \sum_{j|(i,j)\in T} f_k(X_{ij}) - \frac{1}{\hat{m}_i} \sum_{j|(i,j)\in Q} \mathbb{E}_{p(X_{ij}|u_i,v_j)}[f_k(X_{ij})] \tag{6.7}$$

[12] These functions may also be measured relationships between the users and items, in which case the required assumption is that the latent relationship between any user and item deterministically determines the measurable multi-relations between them.

between the average of $f_k(X) \mapsto [0,1]$ over the observed ratings and the average of the expected value of $f_k(X)$ over the query ratings is bounded above by

$$\epsilon_{ik} \leq \sqrt{\frac{\ln \frac{2}{\delta}}{2m_i}} + \sqrt{\frac{(m_i + \hat{m}_i) \ln \frac{2}{\delta}}{2m_i \hat{m}_i}} \tag{6.8}$$

with probability $1 - \delta$.

The proof is deferred to Appendix B.4. The same difference is also bounded by the following corollary.

Corollary 6.3.2. *The difference ϵ_{ik} defined in Equation (6.7) is bounded below by*

$$\epsilon_{ik} \geq -\sqrt{\frac{\ln \frac{2}{\delta}}{2m_i}} - \sqrt{\frac{(m_i + \hat{m}_i) \ln \frac{2}{\delta}}{2m_i \hat{m}_i}} \tag{6.9}$$

with probability $1 - \delta$.

Since Theorem 6.3.1 holds for any bounded function, applying the result for function $1 - f_k(X)$ proves the corollary. Moreover, the same bounds hold for item ratings as summarized by the following corollary.

Corollary 6.3.3. *For the ratings of item j, the difference between the average of $f_k(X) \mapsto [0,1]$ over the observed ratings and the average of the expected value of $f_k(X)$ over the query ratings is bounded above and below with high probability.*

The proof follows by replacing all references to users with references to items and vice versa. This produces concentration bounds that are similar to those in Equations (6.8) and (6.9).

Not only does Theorem 6.3.1 provide assurance that we should predict distributions with similar averages across training and query entries of X, the dependence of the bounds on the number of training and query samples adaptively determine how much deviation can be allowed between these averages. Due to the linearity of the expectation operator, each bound above produces a linear inequality or half-space constraint on $p(X_{ij}|u_i, v_j)$. The conjunction of all such half-spaces[13] forms a convex hull Δ of allowable choices for the distribution $\prod_{(i,j)\in Q} p(X_{ij}|u_i, v_j)$. These bounds hold

[13]Combining the bounds for each user, item, and feature requires using a significantly looser union bound, which maintains the motivation for the learning algorithm, but is no longer likely to hold in finite data of realistic cardinality.

without parametric assumptions about the conditional probabilities $p(X_{ij}|u_i, v_j)$ generating the ratings. They also make no parametric assumptions about the distributions $p(u)$ and $p(v)$ generating the users and the items. The δ confidence value adjusts all the deviation inequalities and can be used as a regularization parameter. A small value of δ effectively relaxes the convex hull of inequalities while a large value of δ permits less deviation and shrinks the hull. Thus, δ controls all deviation inequalities which also individually depend on the cardinality of their training and query ratings.

The maximum entropy method for estimating rating probabilities All probability distributions must be members of the simplex, denoted Δ. To choose from the candidate distributions $p \in \Delta$ that fit the constraints derived in the previous section (and reside inside the prescribed convex hull), we apply the maximum entropy method. The solution distribution p recovered will be of the form $\prod_{(i,j)\in Q} p(X_{ij}|u_i, v_j)$. We choose the distribution that contains the least information subject to the deviation constraints from the training data. Alternatively, we can minimize relative entropy to a prior p_0 subject to the constraints defined by the deviation bounds. This is a strictly more general approach since, when p_0 is uniform, minimum relative entropy coincides with standard maximum entropy. We suggest using a single identical maximum likelihood multinomial distribution over all ratings independent of user and item for the prior, namely $\prod_{(i,j)\in Q} p_0(X_{ij})$. Hence, we use the terms maximum entropy and minimum relative entropy interchangeably.

Assume we are given a set of functions F where $f_k \in F$ and $k \in \{1, \ldots, |F|\}$. Let α_i be the maximum deviation allowed for each function's average expected value for i. Let β_j be the maximum deviation allowed for each function's average expected value for item j. The α and β ranges are set according to Theorem 6.3.1 and its corollaries. For some δ, the allowed deviations are

$$\begin{aligned} \alpha_i &= \sqrt{\frac{\ln \frac{2}{\delta}}{2m_i}} + \sqrt{\frac{(m_i + \hat{m}_i) \ln \frac{2}{\delta}}{2m_i\hat{m}_i}} \\ \beta_j &= \sqrt{\frac{\ln \frac{2}{\delta}}{2n_j}} + \sqrt{\frac{(n_j + \hat{n}_j) \ln \frac{2}{\delta}}{2n_j\hat{n}_j}}. \end{aligned}$$

These scalars summarize the convex hull Δ of distributions that are structured according to the prescribed sampling hierarchy.

The primal maximum entropy problem is

$$\begin{aligned}
\max_{p} \; & \sum_{ij \in Q} H(p_{ij}(X_{ij})) + \sum_{ij \in Q, X_{ij}} p_{ij}(X_{ij}) \ln p_0(X_{ij}) \\
\text{s.t.} \; & \left| \frac{1}{\hat{m}_i} \sum_{j | ij \in Q} \sum_{X_{ij}} p_{ij}(X_{ij}) f_k(X_{ij}) - \mu_{ik} \right| \leq \alpha_i, \forall i, k \\
& \left| \frac{1}{\hat{n}_j} \sum_{i | ij \in Q} \sum_{X_{ij}} p_{ij}(X_{ij}) f_k(X_{ij}) - \nu_{jk} \right| \leq \beta_j, \forall j, k.
\end{aligned}$$

In the above equation, $p_{ij}(X_{ij})$ is used as shorthand for the conditional probabilities $p(X_{ij}|u_i, v_j)$ for space reasons. In practice, the dual form of the problem is solved. This is advantageous because the number of queries is typically $\mathcal{O}(mn)$, for m users and n items, whereas the number of constraints is $\mathcal{O}(m + n)$. Moreover, only a sparse set of the constraints is typically active. Since the primal problem is more intuitive, the details of the dual formulation are deferred to Appendix C.1. Given a choice of the feature functions F, a setting of δ and a set of observations, it is now straightforward to solve the above maximum entropy problem and obtain a solution distribution $\prod_{(i,j) \in Q} p(X_{ij}|u_i, v_j)$.

Feature functions This subsection specifies some possible choices for the set of feature functions F. These are provided only for illustrative purposes since many other choices are possible as long as the functions have $[0, 1]$ range. For discrete ratings $\{1, \ldots, K\}$, a plausible choice of F is the set of all possible conjunctions over the K settings. For example, the set of all singleton indicator functions and the set of pairwise conjunctions encodes a reasonable set of features when K is small:

$$\begin{aligned}
f_i(x) &= I(x = i), & i \in \{1, \ldots, K\}, \\
f_{i,j}(x) &= I(x = i \vee x = j), & (i, j) \in \{1, \ldots, K\}^2.
\end{aligned}$$

These are used to populate the set F as well as the linear and quadratic transformation functions $f(x) = (x - 1)/(K - 1)$ and $f(x) = (x - 1)^2/(K - 1)^2$. Each of these functions is bounded by $[0, 1]$. It is thus possible to directly apply Theorem 6.3.1 and produce the constraints for the maximum entropy problem.

While previous ℓ_1-regularized maxent methods have combined the positive and negative absolute value Lagrange multipliers [Dudík *et al.*, 2007], we found that on our data, this led to numerical issues during optimization. Instead, we optimize both the positive and negative multipliers even

though only one will be active at the solution. To reduce computation time, we use a simple cutting plane procedure. We initialize the problem at the prior distribution where all Lagrange multipliers are set to zero and find the worst violated constraints. We solve the dual (Appendix C.1), fixing all Lagrange multipliers at zero except the most violated constraints, and continue increasing the constraint set until all primal constraints are satisfied. In the worst case, this method eventually must solve a problem with half of the Lagrange multipliers active. Typically, we only need to optimize a much smaller subset. We solve the optimizations using the LBFGS-b optimizer [Zhu *et al.*, 1997].

6.3.2 Experimental evaluation of collaborative filtering via rating concentration

This section compares the maximum entropy (maxent) method against other approaches for both synthetic and real data sets. A popular contender method is *fast max-margin matrix factorization* (fMMMF) [Srebro *et al.*, 2004] which factorizes the rating matrix subject to a low trace-norm prior. One variant of MMMF uses logistic loss to penalize training rating errors, which provides a smooth approximation of hinge-loss. The cost function for fast MMMF with all-threshold logistic loss is as follows [Rennie, 2007]:

$$||U||_{\mathrm{F}}^2 + ||V||_{\mathrm{F}}^2 + C \sum_{r,(i,j)\in T} \ln\left(1 + e^{\mathrm{sgn}(X_{ij}-r)(\theta_{ir}-u_i^\top v_j)}\right).$$

The sgn function outputs $+1$ when the input is positive and -1 otherwise. Consider a probabilistic interpretation of fMMMF where the above cost function is viewed as a log-loss associated with the likelihood

$$p(X, U, V|\theta) = \prod_{(i,j)\in T} p(X_{ij}|u_i, v_j, \theta) \prod_i p(u_i) \prod_j p(v_j).$$

The priors $p(u_i)$ and $p(v_j)$ on the user and item descriptors are zero-mean, spherical Gaussians scaled by $\frac{1}{C}$ and the conditional rating probability is defined by

$$p(X_{ij}|u_i, v_j, \theta) \propto \prod_r \frac{1}{1 + e^{\left(\mathrm{sgn}(X_{ij}-r)(\theta_{ir}-u_i^\top v_j)\right)}}. \tag{6.10}$$

The above formula allows direct comparison of log-likelihood performance of distributions estimated via Equation (6.10) versus the proposed maximum entropy method. The logistic-loss Fast MMMF method is evaluated using the author's publicly available code [Rennie, 2007].

Two additional comparison methods were considered: the *probabilistic matrix factorization* (PMF) technique [Salakhutdinov and Mnih, 2008b] and its Bayesian extension [Salakhutdinov and Mnih, 2008a]. Both methods learn the parameters of the graphical model structure in Figure 6.2. However, each makes parametric assumptions: Gaussian observation noise and Gaussian priors on the user and item descriptors. PMF performs *maximum a posteriori* (MAP) estimation on the model and Bayesian PMF uses Gibbs sampling to simulate integration over the Gaussian priors. Since PMF only estimates Gaussian means, the noise parameters can be set subsequently by choosing the value that produces the highest likelihood.

Table 6.2: Average likelihood and divergence results for synthetic data. Values are averages over 10 folds and statistically significant improvements (according to a two-sample t-test) are displayed in bold. Higher log-likelihood and lower KL-divergence are better.

	fMMMF	Maxent
Log-Likelihood	-39690 ± 214	$\mathbf{-35732 \pm 216}$
KL-divergence	11254 ± 315	$\mathbf{4954 \pm 154}$

Finally, for experiments with real data, we also compare against the likelihoods using simple estimators such as a uniform distribution over all ratings or an identical maximum likelihood estimate $p_0(X_{ij})$ for all query ratings.

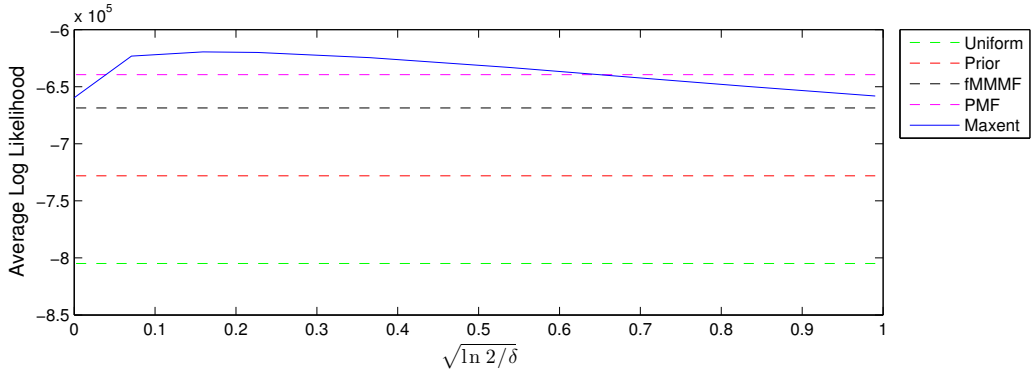


Figure 6.3: Average log likelihoods for each algorithm on Movielens data. The log-likelihoods are plotted against the regularization or confidence parameter δ in the maximum entropy method.

Synthetic experiments One drawback of real data experiments is that ground truth rating distributions are not given, only samples from these are available. Therefore, consider a synthetic scenario where the exact rating distributions are specified and used to generate samples to populate the rating matrix.

First, 500 users and 500 items were sampled from a uniform probability density such that $u_i, v_j \in [0, 1]^5$. The rating distribution is then a multinomial with entries proportional to the element-wise product of the corresponding user-item pair: $p(X_{ij} = r|u_i, v_j) \propto u_i(r)v_j(r)$. Subsequently, a random subset T of training ratings was formed by drawing one sample from 20% of the entries of the rating matrix. These observed rating samples were provided to both fMMMF and maxent. For both algorithms, 10% of the training set was used for cross-validation. The appropriate scalar regularization parameter (C or δ) was chosen by maximizing likelihood on this validation set.

A random subset Q of testing ratings was then formed by drawing one sample from 20% of the entries of the rating matrix (these entries were disjoint from the training entries). The out-of-sample test likelihood was then computed over Q. This setting is similar to a typical testing procedure with real data. Higher likelihood scores indicate a better estimate of the rating distribution however each rating distribution is only sampled once. Therefore, we also report the Kullback-Leibler (KL) divergence between the true query multinomials and the estimated distributions $p(X_{ij}|u_i, v_j)$. The above experiment is repeated ten times and average results are reported across the trials. The maximum entropy method obtains higher likelihood scores as well as lower divergence from the true distribution than the fMMMF method. The advantages are statistically significant under both performance metrics. Table 6.2 summarizes the synthetic experiments.

Movie ratings This section compares several algorithms on the the popular Movielens data set. This data set is a collection of over one million ratings from over six thousand users for over three thousand movies. The ratings are integers ranging from 1 to 5. We randomly split the ratings in half to define the training and query user-item pairs. To choose regularization parameters, 20% of the training ratings were held out as a validation set. Finally, we test the resulting likelihoods on the query ratings.

On three random splits of training/validation/testing sets, the log-likelihood of the testing ratings was obtained for several methods. The maximum entropy method obtains the highest test like-

Table 6.3: Query rating log-likelihoods on Movielens data. The maximum entropy (maxent) method has statistically significantly higher average likelihood over the three trials according to a two-sample t-test with p value of as little as $1\text{e}-5$. We convert the threshold model of fMMMF to a distribution for this comparison.

	Uniform	Prior	fMMMF Distrib.	PMF	Maxent
Split 1	-8.0489e+05	-7.2800e+05	-6.6907e+05	-6.3904e+05	-6.1952e+05
Split 2	-8.0489e+05	-7.2796e+05	-6.6859e+05	-6.3936e+05	-6.1977e+05
Split 3	-8.0489e+05	-7.2809e+05	-6.6819e+05	-6.3987e+05	-6.1931e+05
Average	-8.0489e+05	-7.2802e+05	-6.6862e+05	-6.3942e+05	**-6.1953e+05**

lihood, which improves over 20% more than the improvement obtained by the leading contender method relative to the naive $p_0(x)$ prior. Figure 2 illustrates the average likelihood for various regularization parameter settings compared to the competing methods. Our likelihood improvement is statistically significant according to a two-sample t-test with the rejection threshold below $1\text{e}-5$. Log-likelihoods of each method are listed in Table 6.3.

We also compare ℓ_2 error on the ratings. This is done by recovering point-estimates for the ratings by taking the expected value of the rating distributions. Comparing against other maximum-likelihood methods like fMMMF and PMF, maxent obtains slightly higher accuracy. All three methods are surpassed by the performance of Bayesian PMF, however. Interestingly, simply averaging the predictions of fMMMF and maxent or the predictions of PMF and maxent produces more accurate predictions than either algorithm alone. This suggests that the methods are complementary and address different aspects of the collaborative filtering problem. The ℓ_2 errors are listed in Table 6.4.

6.3.3 Discussion

A method for collaborative filtering was provided that exploits concentration guarantees for functions of ratings. The method makes minimal assumptions about the sampling structure while otherwise remaining agnostic about the parametric form of the generative model. By assuming that users and items are sampled *iid*, general concentration inequalities on feature functions of the ratings were

Table 6.4: Root mean square or ℓ_2 performance of various algorithms. Maxent gives the least error among the MAP methods (fMMMF, PMF and maxent) but Bayesian PMF outperforms all methods. Combining maxent with other MAP methods improves accuracy.

	fMMMF	PMF	Maxent	BPMF	Maxent+fMMMF	Maxent+PMF
Split 1	0.9585	0.9166	0.9168	0.8717	0.9079	0.8963
Split 2	0.9559	0.9175	0.9162	0.8710	0.9052	0.8965
Split 3	0.9583	0.9186	0.9166	0.8723	0.9065	0.8973
Average	0.9575	0.9176	0.9165	0.8717	0.9065	0.8967

obtained. A solution probability distribution constrained by these concentration inequalities was obtained via the maximum entropy criterion. This method produces state-of-the-art performance by exploiting different intuitions and simpler assumptions than leading contenders. Furthermore, the proposed method is complementary with the assumptions in other approaches. Simply exploiting concentration constraints produces strong collaborative filtering results and more sophisticated models may also benefit from such bounds.

6.4 Summary of degree-based approaches to collaborative filtering

This section addresses the collaborative filtering problem. Initially a simple, binary instance of collaborative filtering is posed as a graph estimation problem, where an algorithm is derived to post-process a standard collaborative filtering technique to obey degree-based arguments. The degree-based intuition is that, under a latent space sampling assumption, the rate of relations predicted in testing data should be concentrated toward the empirical rate of relations in training data. These degree-based arguments are then extended to a multi-rating, or multi-relational setting, where they are used to estimate probabilities of ratings. The sampling assumptions in this chapter are equivalent to the latent-space generative model for graphs, but by focusing only on the degree, or counting statistics, the post-processing procedure can be used on any parameterization of the latent space, and the maximum entropy estimation procedure performs estimation with an agnostic parameterization of the latent space.

While the degree-based arguments in this chapter are engineered for the particular task, leading to given degree priors, the next chapter addresses the case where the degree preferences are not known and should be learned from observed data.

Chapter 7

Learning from and Predicting Networks

The preceding chapters have primarily handled the inference task with a given degree-sensitive probability distribution. This chapter addresses the task of learning such a distribution from observed data. The chapter begins with a motivational example demonstrating the utility of degree information in modeling a synthetic network generated using the *preferential attachment* model. Rather than being given a distribution directly, this example showcases the *maximum a posteriori* technique when the unseen, latent graph structure generates data, and the prediction algorithm infers the structure from the data observations in conjunction with degree priors. In this initial motivational example, the degree priors and data generative model are given. The later sections of the chapter describe an approach for simultaneously learning both the degree preference functions and the edge data likelihood functions by posing the problem as *structured metric learning*.

The ideas discussed in this chapter build off of the background of Chapter 2, but are primarily focused on leveraging node information in addition to network structure for learning and prediction, as well as the application of degree information for the prediction task. As in prior studies, the task of link prediction is to predict unobserved connections between nodes using models learned from observed connectivity. The next few sections discuss approaches to this prediction task while explicitly modeling node degrees and using degree-based subgraph estimation as a prediction function.

7.1 Exact degree priors in synthetic scale-free networks

This section describes a synthetic experiment testing the benefit of degree information for estimating a graph with a realistic degree distribution in the presence of observation noise. Using noisy observations of graph edges, the target is to reconstruct the true graph. By comparing estimation using degree information with estimation with no degree information, we seek insight into what gains are made by using degrees in prediction.

We generate a graph using a preferential attachment model [Albert and Barabási, 2002], which produces the power-law degree distributions seen in natural networks. This model generates graphs with few high-degree nodes and many low-degree nodes, where the degree distribution has a power-law decay toward higher degrees. Another important feature of preferential-attachment graphs is that they are extremely sparse. We generate the graph of 500 nodes by first initializing two connected nodes. We then grow the graph by adding nodes one at a time, connecting new incoming nodes to a constant m of the current nodes in the graph, where the node to connect to is selected with probability proportional to its current degree. This produces a "rich-get-richer" effect and generates a graph with supposedly realistic structural properties. For the more realistic, very sparse graph, we use $m = 1$, and to generate a less realistic, dense graph, we set m to 200.

Using each generated graph, we generate observations X_{ij} for each pair of nodes by sampling from a one-dimensional normal distribution with a mean at 1.0 if an edge exists between the node pair and a mean at 0.0 if no edge is present. These observations are noisy indicators of edge presence. The observation X_{ij} is sampled as

$$X_{ij} \in \begin{cases} \mathcal{N}(1.0, \sigma) & \text{if } A_{ij} = 1, \\ \mathcal{N}(0.0, \sigma) & \text{if } A_{ij} = 0. \end{cases}$$

We assume the variance and the conditional means are known. A simple estimation procedure with minimal degree information is to compare the likelihood of the observed data for each of the two possible means and estimate based on which likelihood is greater, incorporating a prior likelihood $\Pr(A_{ij}) = |E|/(|V|(|V|-1))$ for edge appearance estimated from the true graph. We refer to this technique as independent MAP estimation using a simple prior, expressed as

$$\operatorname*{argmax}_{A} \prod_{ij} \Pr(X_{ij}|A_{ij}) \Pr(A_{ij}).$$

An intermediary technique, which somewhat preserves the known sparsity of the target graph, is to select only the $|E|$ most likely edges. The intermediary technique can be expressed as

$$\operatorname*{argmax}_{A} \prod_{ij} \Pr(X_{ij}|A_{ij}) \Pr(A),$$

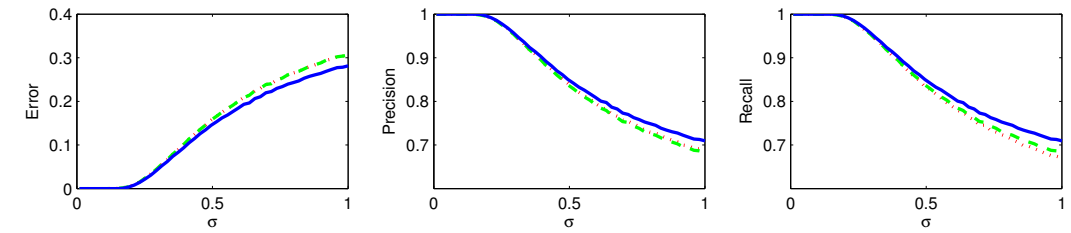
where $\Pr(A)$ is a global prior on A that puts uniform likelihood on graphs with $|E|$ edges and zero probability on other graphs. Finally, the most structure-preserving technique we compare is to find the maximum likelihood b-matching using the true degrees of the nodes.

The results on the realistic, sparse graph show that as the noise parameter increases, performance decreases for all three techniques. Nonetheless, until a critical noise level is reached, at which point all methods fail, using b-matching with the exact degrees produces lower zero-one error than the less structured approaches. Figure 7.1 contains plots of the error, the precision and the recall on the edge variables. Note that the precision of the independent MAP estimate actually improves as the noise parameter increases because it is in fact estimating that all but the few most certain edges are off, providing good precision, while the other two methods must estimate that some edges are on. Naturally, in a practical setting, a prediction of a completely disconnected graph cannot possibly be useful, so we consider this behavior pathological. This is why we also compare the three techniques on the dense graph, where 0.4831 of possible edges are active. On the dense graph, b-matching is always more accurate according to all three metrics for all noise levels.

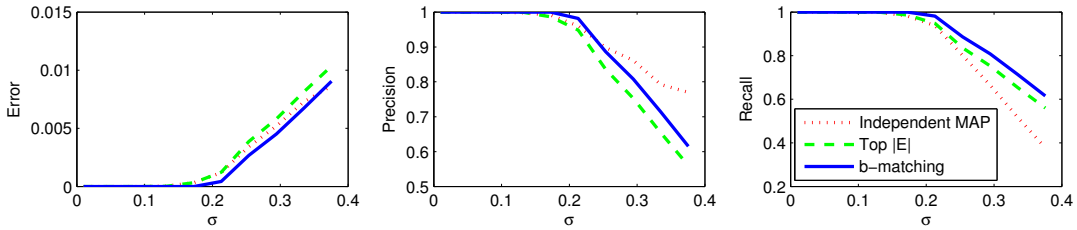
Finally, it is important to note that, while measurements in this experiment focus on accuracy, there are settings where the structure is not just a tool to improve estimation accuracy but a necessity. In those settings, the independent or the intermediary approaches may not even yield useable estimates.

7.2 Structured metric learning approaches for graph prediction

Most of the models and methods discussed so far in this thesis either expect that the graph distribution is known or that the an estimate of the graph distribution is engineered. In some scenarios, it is natural to devise a model for edge likelihood and degree preferences. In many other scenarios, however, the best edge potentials and degree preferences are unknown and difficult to design. In these scenarios, it is useful to have a learning algorithm that can automatically determine what edge



(a) Dense graph



(b) Sparse graph

Figure 7.1: Error and accuracy measures of synthetic graph reconstruction. Independent MAP is using a single prior for all edges, Top $|E|$ is estimating that the $|E|$ most likely edges are on, regardless of what nodes they connect, and b-matching uses the true degrees. The plots from left to right are error, where lower is better, precision, where higher is better, and recall, where higher is better. The horizontal axis is the Gaussian noise parameter σ used to sample the observed data.

weights and degree preferences most genuinely represent relevant data. This section covers an approach to learning model parameters that allows for generalized graph structure prediction in new data. The techniques described here learn a model from node data and labeled graph connectivity, which generalizes to new, previously unseen data.

A natural technique for determining edge likelihoods is to parameterize the likelihood with a similarity between nodes. Under the assumption that some measure of similarity is strongly correlated with connectivity, a concept often referred to as *homophily* in social network analysis [Christakis and Fowler, 2011], ideas from *metric learning* can be adopted to the task of graph prediction. Given descriptions of two nodes, some measure of similarity should indicate how likely they are to be connected in a network.

Empirical evidence in the previous section indicates a significant advantage when using degrees in conjunction with edge likelihoods in predicting graph structure. However, in more realistic testing scenarios, the target degrees of query nodes is unknown. The degree distribution for some nodes may be non-stationary and depend on their attributes, particularly if such attributes contain information representing the capacity for edges of that node. For example, in the LinkedIn online social network, an individual whose job occupation is "Marketing Agent" is likely to have more connections than an individual whose occupation is "Mathematician".

Combining the ideas that node similarity relates to edge likelihood and that degree preference is dependent on node attributes yields a learnable, conditional model for graph structure likelihood conditioned on node features. The learning task is referred to as *degree distributional metric learning* (DDML) [Shaw *et al.*, To appear]. The sections below describe such the degree distributional metric model, learning algorithms for the model parameters, variants and experiments using DDML.

Related Work Metric learning is often applied to supervised problems such as classification [Chechik *et al.*, 2010; Weinberger and Saul, 2009]. These methods first build a k-nearest neighbors graph from training data and then learn a Mahalanobis distance metric keeping points with the same label close while pushing away class impostors, pairs of points which are connected but of a different class. Like these previous metric learning methods, DDML aims to learn a metric that, when combined with degree preferences, produces a graph that matches as closely as possible with the original input connectivity structure. Instead of pushing away class impostors, DDML pushes

away *graph impostors*, disconnected points that are close in distance. Moreover, traditional metric learning algorithms impose no requirements on the degree distribution of the connectivity matrix found under the learned metric, whereas DDML ensures the degree distribution of the induced connectivity closely matches that of the true connectivity.

The training phase of the DDML algorithm is similar to structure preserving embedding (SPE) [Shaw and Jebara, 2009]. SPE learns coordinates for nodes in a graph such that applying a connectivity algorithm to those node coordinates yields the original adjacency matrix. Where SPE learns an embedding explicitly from only the target adjacency matrix, DDML learns a metric corresponding to a linear transformation of additional node features given as input. Furthermore, SPE constrains the exact degrees of the nodes in the input graph, DDML learns more general degree preference functions based on the attributes of the nodes. This additional mapping between features and degrees allows the learned degree distribution metric to be applied to new, unseen graphs. Like SPE, DDML can use cutting-plane approach similar to that of structured prediction, and is thus strongly related to results by [Caetano *et al.*, 2009] wherein the weights of a linear assignment problem, *i.e.*, maximum weight bipartite matching, are learned to closely mimic a quadratic assignment problem. Their formulation as a maximum margin empirical risk minimization also results in a quadratic optimization similar in form to structural support vector machines [Joachims *et al.*, 2009].

The stochastic version of the DDML algorithm is related to the PEGASOS support vector machine (SVM) algorithm [Shalev-Shwartz *et al.*, 2007], which, through the use of stochastic optimization, solves the SVM optimization in time independent of the number of data points in the input.

7.3 Degree distributional metric learning

Degree distributional metric learning (DDML) aims to simultaneously learn similarity and degree preference functions that accurately predict graph structure from node descriptor data [Shaw *et al.*, To appear]. Given as input an adjacency matrix $\mathbf{A} \in \mathbb{B}^{n \times n}$, and node features $\mathbf{X} \in \mathbb{R}^{d \times n}$, DDML learns a similarity function $f : \{\mathbb{R}^D, \mathbb{R}^D\} \mapsto \mathbb{R}$ that takes two vectors as input and outputs a real value, and a degree preference function $g : \{\mathbb{R}^D, \mathbb{N}\} \mapsto \mathbb{R}$, which takes a node descriptor and a candidate degree d and outputs a real valued preference score for that node having degree d.

The degree distributional metric is parameterized by matrices $\mathbf{M} \in \mathbb{R}^{D \times D}$ and $\mathbf{S} \in \mathbb{R}^{n \times D}$. The distance between two points under the metric is defined as $D_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i - \mathbf{x}_j)^\top \mathbf{M}(\mathbf{x}_i - \mathbf{x}_j)$. When the metric is the identity $\mathbf{M} = I_d$, $D_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j)$ represents the squared Euclidean distance between the i'th and j'th points. Using the notation that $\mathbf{s}_c$ is the $1 \times D$ dimensional c'th row of $\mathbf{S}$,

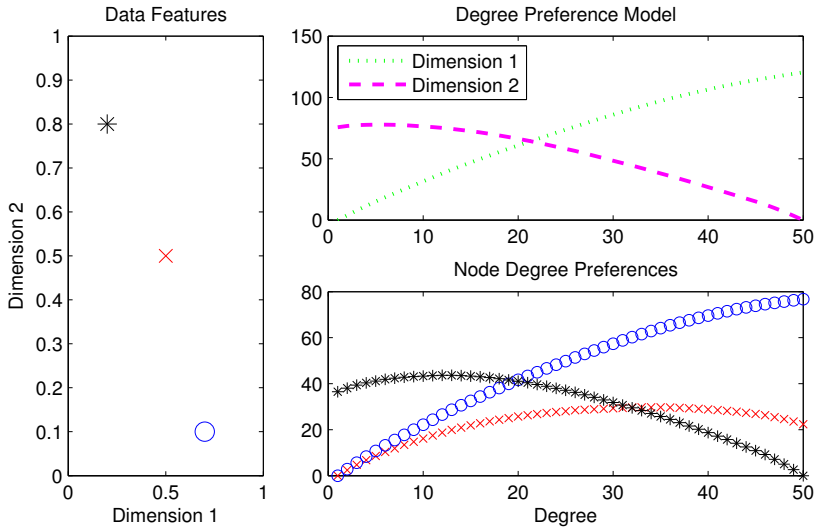


Figure 7.2: Example of non-stationary degree preference functions. This image illustrates the degree preference functions of three different nodes in $\mathbb{R}^2$. The descriptors of the nodes are shown on the left. Top right: The curves are the cumulative sum of the degree preference parameter $\mathbf{S}$ along each dimension. These curves are the bases for nodes' degree preference functions. Bottom right: The resulting degree preference functions, each of which is a linear combination of the bases above. For example, the blue circle node has a high value on Dimension 1 and a low value on Dimension 2, so its resulting degree preference is skewed toward high degrees like the basis for Dimension 2.

the degree preference function is

$$g(\mathbf{x}_i, b; \mathbf{S}) = \sum_{c=1}^{b} \mathbf{s}_c \mathbf{x}_i.$$

This linear function of b can be viewed as a weighted sum of degree preference functions, defined along the columns of $\mathbf{S}$, scaled by the dimensions of $\mathbf{x}_i$. Figure 7.2 illustrates a simple two-dimensional example of degree preferences resulting from points in different parts of the input space.

A graph is predicted by finding a connectivity that maximizes the sum of the similarity between neighbors and the degree preference functions for in-degree and out-degree of each node. The prediction operation computes

$$\underset{\mathbf{A}}{\operatorname{argmax}} \qquad F(\mathbf{A}|\mathbf{X}, \mathbf{M}, \mathbf{S}), \tag{7.1}$$

where

$$F(\mathbf{A}|\mathbf{X}, \mathbf{M}, \mathbf{S}) = -\sum_{ij|A_{ij}=1} D_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j; \mathbf{M}) + \sum_i g\left(\mathbf{x}_i, \sum_j A_{ij}; \mathbf{S}\right).$$

Prediction function (7.1) is a parameterized form of the degree-based subgraph estimation objective (3.1), which can be efficiently optimized by the procedure in Chapter 3.

7.3.1 Structured prediction learning

Given input training data, a learning algorithm aims to fit the parameters $\mathbf{M}$ and $\mathbf{S}$ to minimize a regularization term and empirical risk according to a loss function between the labeled graph connectivities and the predicted connectivities. We use the normalized Hamming distance for the loss function

$$\Delta(\mathbf{A}, \tilde{\mathbf{A}}) = \sum_{ij|A_{ij} \neq \tilde{A}_{ij}} \frac{1}{n^2 - n},$$

which is the proportion of possible edges (assuming no self-loops) that are misclassified. Using the Frobenius ℓ_2-norm of the parameter matrices as a regularization term, learning is done by solving

$$\begin{aligned} \min_{\mathbf{M},\mathbf{S},\xi} \quad & \frac{\lambda}{2}\left(||\mathbf{M}||_{\text{Fro}}^2 + ||\mathbf{S}||_{\text{Fro}}^2\right) + \xi \\ \text{s.t.} \quad & \left[F(\mathbf{A}|\mathbf{X}, \mathbf{M}, \mathbf{S}) - F(\tilde{\mathbf{A}}|\mathbf{X}, \mathbf{M}, \mathbf{S})\right] \geq \Delta(\mathbf{A}, \tilde{\mathbf{A}}) - \xi, \forall \tilde{\mathbf{A}}. \end{aligned} \tag{7.2}$$

We can optionally add a concavity requirement if the values in the data vectors are non-negative. Without this constraint, the concavity of the degree preference functions is implicitly enforced. For non-negative data, the concavity of function $g(\cdot)$ is maintained by enforcing that, for $1 \leq i \leq n-1, 1 \leq j \leq d$,

$$S_{i,j} \geq S_{i+1,j}. \tag{7.3}$$

The optimization is a quadratic program with exponentially many linear constraints, and is, with the exception of the concavity constraints, of the same form as a structural support vector machine

(SVM) [Joachims *et al.*, 2009]. Thus, the established cutting-plane approach (with known efficiency guarantees) can be applied. The solution to (7.2) is found by maintaining a working set of constraints, solving for the optimal $\mathbf{M}$ and $\mathbf{S}$, then adding the worst-violated constraint by the current solution and repeating. Borrowing the terminology from structural SVM, the procedure that finds the worst violated constraint is called the *separation oracle*.

Algorithm 5 Degree distributional metric learning with cutting-plane optimization.

Require: Labeled graph $\{\mathbf{X}, \mathbf{A}\}$, regularization parameter λ, and stopping tolerance ϵ

1: Initialize $\mathbf{M}, \mathbf{S}$ $\{$*e.g.*, $\mathbf{M} \leftarrow \mathbf{I}$ and $\mathbf{S} \leftarrow [0]\}$

2: Constraint set $\mathcal{C} \leftarrow \emptyset$ $\{$or optionally add concavity constraints from Eq. (7.3)$\}$

3: **repeat**

4: $\quad (\mathbf{M}, \mathbf{S}, \xi) \leftarrow \operatorname{argmin}_{\mathbf{M},\mathbf{S},\xi \geq 0} \frac{\lambda}{2} \left(||\mathbf{M}||^2_{\text{Fro}} + ||\mathbf{S}||^2_{\text{Fro}} \right) + \xi$ s.t. $\mathcal{C}$

5: $\quad$ (Optional) Project $\mathbf{M}$ onto PSD cone

6: $\quad \tilde{\mathbf{A}} \leftarrow \operatorname{argmax}_{\mathbf{A}} F(\mathbf{A}|\mathbf{X}, \mathbf{M}, \mathbf{S}) + \Delta(\mathbf{A}, \mathbf{A})$

7: $\quad \mathcal{C} \leftarrow \mathcal{C} \cup \left[F(\mathbf{A}|\mathbf{X}, \mathbf{M}, \mathbf{S}) - F(\tilde{\mathbf{A}}|\mathbf{X}, \mathbf{M}, \mathbf{S}) \right] \geq \Delta(\mathbf{A}, \tilde{\mathbf{A}}) - \xi$

8: **until** $\Delta(\mathbf{A}, \tilde{\mathbf{A}}) - \xi - \left[F(\mathbf{A}|\mathbf{X}, \mathbf{M}, \mathbf{S}) - F(\tilde{\mathbf{A}}|\mathbf{X}, \mathbf{M}, \mathbf{S}) \right] < \epsilon$

Separation Oracle Since the constraint involves independent candidate adjacency matrices, each worst-violator $\tilde{\mathbf{A}}$ can be found independently using the following maximization which is almost identical to the prediction function in Eq. 7.1,

$$\tilde{\mathbf{A}} \quad = \quad \operatorname*{argmax}_{\mathbf{A}} F(\mathbf{A}|\mathbf{X}, \mathbf{M}, \mathbf{S}) + \Delta(\mathbf{A}, \mathbf{A}).$$

The above can be maximized by adding the decomposed loss to the primary edge weights of the b-matching input. Specifically, for edge (i, j), let edge weight $w(x_i, x_j)$ be

$$w(x_i, x_j) = \begin{cases} f(\mathbf{x}_i, \mathbf{x}_j; \mathbf{M}) - \frac{1}{(n^2 - n)} & \text{if } A_{ij} = 1, \\ f(\mathbf{x}_i, \mathbf{x}_j; \mathbf{M}) + \frac{1}{(n^2 - n)} & \text{if } A_{ij} = 0. \end{cases}$$

Essentially, adding these loss terms rewards the separation oracle for selecting connectivities that are both high weight and high loss.

Constraints on $\mathbf{M}$ and $\mathbf{S}$ A useful tool for computing distances is the linear transform

$$D_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top \mathbf{M} \mathbf{x}_i + \mathbf{x}_j^\top \mathbf{M} \mathbf{x}_j - \mathbf{x}_i^\top \mathbf{M} \mathbf{x}_j - \mathbf{x}_j^\top \mathbf{M} \mathbf{x}_i, \tag{7.4}$$

which allows linear constraints on the distances to be written as linear constraints on the $\mathbf{M}$ matrix. For notational cleanliness, let the degrees of all nodes be stored in array c, such that the degree in $\mathbf{A}$ of the i'th node is $c[i] = \sum_j A_{ij}$, and let the degree in $\tilde{\mathbf{A}}$ be stored in $\tilde{c}$. For any false graph $\tilde{\mathbf{A}}$, the linear constraint on $\mathbf{M}$ and $\mathbf{S}$ is

$$\begin{aligned}\Delta(\mathbf{A}, \tilde{\mathbf{A}}) - \xi \quad \leq \quad & \operatorname{tr}\left(\mathbf{M}^\top \left(\sum_{ij}(A_{ij} - \tilde{A}_{ij})\left(\mathbf{x}_i\mathbf{x}_i^\top + \mathbf{x}_j\mathbf{x}_j^\top - \mathbf{x}_i\mathbf{x}_j^\top - \mathbf{x}_j\mathbf{x}_i^\top\right)\right)\right) \\ & + \operatorname{tr}\left(\mathbf{S}^\top \left(\sum_i \sum_{d=\tilde{c}[i]}^{c[i]} \delta_d \mathbf{x}_i^\top - \sum_{d=c[i]}^{\tilde{c}[i]} \delta_d \mathbf{x}_i^\top\right)\right), \end{aligned} \tag{7.5}$$

where each δ_d is an $n \times 1$ vector of all zeros except in the d'th entry, and is thus used to construct the appropriate matrix coefficient for the $\mathbf{S}$ matrix.

Analysis of structured SVM learning A cutting-plane implementation of structural SVM using the objective, separation oracle, and constraints detailed in the previous few pages optimizes Equation 7.2 by maintaining a working set of constraints, and at each iteration adding the worst-violating constraint from the separation oracle. This optimization procedure is summarized in Algorithm 5. Each iteration is at least the cost of the separation oracle, which is $\mathcal{O}(n^3)$. Thus, the running time of the algorithm is prohibitive for large graphs. The next section describes a stochastic variant of the learning algorithm that solves a relaxation of the objective, but whose running time does not depend on the size of the graph.

7.3.2 Stochastic large-scale learning

Unfortunately, the structured SVM approach for DDML requires iteratively solving maximum weight b-matchings, which becomes expensive in larger graphs. A stochastic learning strategy allows learning of large-scale graphs, and, by parameterizing the degree preference function only up to a fixed maximum degree, also eliminates the dependence of the running time on the size of the graph.

A DDML objective can be written in terms of triplets of nodes i, neighbor j, disconnected node triplets k. The set of all triplets is

$$T = \{(i, j, k) | A_{ij} = 1 \wedge A_{ik} = 0\}.$$

Let $\mathbf{A}^{(i,j,k)}$ denote the false graph produced by toggling the edge between nodes i and j and the edge between nodes i and k. The DDML objective using the triplet-style constraints is

$$\min_{\mathbf{M},\mathbf{S}} L = \frac{\lambda}{2}||\mathbf{M}||^2 - \frac{1}{|T|}\sum_{(i,j,k)\in T} \max(F(\mathbf{A}|\mathbf{X};\mathbf{M},\mathbf{S}) - F(\mathbf{A}^{(i,j,k)}|\mathbf{X};\mathbf{M},\mathbf{S}) + 1, 0). \quad (7.6)$$

The difference in scores decomposes into four scalar values, since the only differences between $\mathbf{A}$ to $\mathbf{A}^{(i,j,k)}$ are that $\mathbf{A}^{(i,j,k)}$ is missing edge (i,j), gains edge (i,k), the degree of node j decreases by one and the degree of node k increases by one. Thus, the difference can be computed by evaluating the distance from node i to node j, the distance from node i to node k, the change in degree preference score from the degree of node j to its degree minus one, and the change in degree preference from the degree of node k from its degree plus one. The difference is then computable as

$$F(\mathbf{A}|\mathbf{X};\mathbf{M},\mathbf{S}) - F(\mathbf{A}^{(i,j,k)}|\mathbf{X};\mathbf{M},\mathbf{S}) = D_{\mathbf{M}}(\mathbf{x}_i,\mathbf{x}_j) - D_{\mathbf{M}}(\mathbf{x}_i,\mathbf{x}_k) + \mathbf{x}_j^\top \mathbf{s}_{(c[j]-1)} - \mathbf{x}_k^\top \mathbf{s}_{(c[k]+1)}.$$

This formulation eliminates the need for the expensive separation oracle and allows stochastic optimization. Using the distance transformation in Equation 7.4, each of the $|T|$ constraints can be written using a sparse matrix $\mathbf{C}^{(i,j,k)}$, where

$$C_{jj}^{(i,j,k)} = 1,\ C_{ik}^{(i,j,k)} = 1,\ , C_{ki}^{(i,j,k)} = 1,\ , C_{ij}^{(i,j,k)} = -1,\ C_{ji}^{(i,j,k)} = -1,\ , C_{kk}^{(i,j,k)} = -1, \quad (7.7)$$

and whose other entries are zero. By construction, sparse matrix multiplication of $\mathbf{C}^{(i,j,k)}$ indexes the proper elements related to nodes i, j, and k, such that $\mathrm{tr}(\mathbf{C}^{(i,j,k)}\mathbf{X}^\top\mathbf{M}\mathbf{X})$ is equal to $D_{\mathbf{M}}(\mathbf{x}_i,\mathbf{x}_j) - D_{\mathbf{M}}(\mathbf{x}_i,\mathbf{x}_k)$. The partial subgradient of L at $\mathbf{M}$ is then

$$\frac{\partial L}{\nabla \mathbf{M}} = \lambda\mathbf{M} + \frac{1}{|T|}\sum_{(i,j,k)\in T_+} \mathbf{X}\mathbf{C}^{(i,j,k)}\mathbf{X}^\top, \quad (7.8)$$

where $T_+ = \{(i,j,k)|F(\mathbf{A}|\mathbf{X};\mathbf{M},\mathbf{S}) - F(\mathbf{A}^{(i,j,k)}|\mathbf{X};\mathbf{M},\mathbf{S}) + 1 > 0\}$. The subgradient with respect to $\mathbf{S}$ is nonzero only along the degrees that change for each triplet, which are at the j'th and k'th nodes,

$$\frac{\partial L}{\nabla \mathbf{S}} = \lambda\mathbf{S} + \frac{1}{|T|}\sum_{(i,j,k)\in T_+} \delta_{(c[j]-1)}x_j^\top + \delta_{(c[k]+1)}x_k^\top. \quad (7.9)$$

To optimize Equation (7.6), we propose using stochastic subgradient descent, by iteratively sampling a random subset of triplets from T and taking a gradient step with a decaying learning rate.

To retain coherence between the different degree functions, we add a requirement that the resulting degree preference function for each node is concave. One way to enforce concavity is by stochastically sampling a node i per iteration, and projecting $\mathbf{S}$ such that entries in $\mathbf{x}_i^\top \mathbf{S}$ are in decreasing order. If further scalability is desired, another strategy is to take a gradient step toward this projection, rather than computing the full projection.

In practice, we optimize this objective function via stochastic subgradient descent. We sample a batch of triplets, replacing T in the objective function with a random subset of T of size B. If a true metric is necessary, we intermittently project $\mathbf{M}$ onto the PSD cone. The pseudocode for stochastic DDML is in Algorithm 6.

Analysis of stochastic DDML Since the stochastic DDML learning algorithm no longer depends on the separation oracle, for this analysis, we omit in this analysis the concavity constraint on g, noting that, in practice, adding projections toward concavity seem to maintain the same convergence behavior derived here. Thus, the stochastic optimization is an instance of the PEGAGOS algorithm [Shalev-Shwartz *et al.*, 2007], albeit a cleverly constructed one. The running time of PEGASOS does not depend on the input size, and instead only scales with the dimensionality, the desired optimization error on the objective function ϵ and the regularization parameter λ. The optimization error ϵ is defined as the difference between the found objective value and the true optimal objective value, $f(\tilde{\mathbf{M}}) - \min_{\mathbf{M}} f(\mathbf{M})$.

Theorem 7.3.1. *Assume that the data is bounded such that* $\max_{(i,j,k)\in T} ||\mathbf{X}\mathbf{C}^{(i,j,k)}\mathbf{X}^\top||_{\mathrm{F}}^2 \leq R$, *and* $R \geq 1$. *During Algorithm 6 at iteration* T, *with* $\lambda \leq 1/4$, *and batch-size* $B = 1$, *let* $\bar{\mathbf{M}} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{M}_t$ *be the average* $\mathbf{M}$ *so far. Then, with probability of at least* $1 - \delta$,

$$f(\bar{\mathbf{M}}) - \min_{\mathbf{M}} f(\mathbf{M}) \leq \frac{84R^2 \ln(T/\delta)}{\lambda T}.$$

Consequently, the number of iterations necessary to reach an optimization error of ϵ *is* $\tilde{O}(\frac{1}{\lambda\epsilon})$.

Proof. The theorem is proven by realizing that, omitting the optional PSD projection step and the concavity projection, Algorithm 5 is an instance of PEGASOS without a projection step on one-class data, since Corollary 2 in [Shalev-Shwartz *et al.*, To appear] proves this same bound for traditional SVM input, also without a projection step. The input to the SVM is the set of all concatenations of the $d \times d$ matrices $\mathbf{X}\mathbf{C}^{(i,j,k)}\mathbf{X}^\top$ and the $\mathbf{S}$ degree parameter matrices as constructed

Algorithm 6 Stochastic degree distributional metric learning using subgradient descent

Require: $\mathbf{A} \in \mathbb{B}^{n \times n}$, $\mathbf{X} \in \mathbb{R}^{d \times n}$, regularization parameter λ, maximum iterations $t_{\max}$, and mini-batch size B

$\mathbf{M}_1 \leftarrow \mathbf{I}_d$, $\mathbf{S}_1 \leftarrow \mathbf{0}_{d,n}$
Compute degree array c s.t. $c[i] = \sum_j A_{ij}, \forall i$
for t from 1 to $t_{\max} - 1$ **do**
  $\eta_t \leftarrow \frac{1}{\lambda t}$
  $\mathbf{C} \leftarrow \mathbf{0}_{n,n}$
  $\mathbf{S}' \leftarrow \lambda \mathbf{S}$
  for b from 1 to B **do**
    $(i, j, k) \leftarrow$ Sample random triplet from $T = \{(i, j, k) \mid A_{ij} = 1, A_{ik} = 0\}$
    if $F(\mathbf{A}|\mathbf{X}; \mathbf{M}_t, \mathbf{S}_t) - F(\mathbf{A}^{(i,j,k)}|\mathbf{X}; \mathbf{M}_t, \mathbf{S}_t) + 1 > 0$ **then**
      $\mathbf{C}_{jj} \leftarrow \mathbf{C}_{jj} + 1$, $\mathbf{C}_{ik} \leftarrow \mathbf{C}_{ik} + 1$, $\mathbf{C}_{ki} \leftarrow \mathbf{C}_{ki} + 1$
      $\mathbf{C}_{ij} \leftarrow \mathbf{C}_{ij} - 1$, $\mathbf{C}_{ji} \leftarrow \mathbf{C}_{ji} - 1$, $\mathbf{C}_{kk} \leftarrow \mathbf{C}_{kk} - 1$
      $\mathbf{s}'_{(c[j]-1)} \leftarrow \mathbf{s}'_{(c[j]-1)} + \mathbf{x}_j$
      $\mathbf{s}'_{(c[k]+1)} \leftarrow \mathbf{s}'_{(c[k]+1)} - \mathbf{x}_k$
    end if
  end for
  $\nabla_t \leftarrow \mathbf{X}\mathbf{C}\mathbf{X}^\top + \lambda \mathbf{M}_t$
  $\mathbf{M}_{t+1} \leftarrow \mathbf{M}_t - \eta_t \nabla_t$
  $\mathbf{S}_{t+1} \leftarrow \mathbf{S}_t - \eta_t \mathbf{S}'$
  $i \leftarrow$ Sample random index
  Project $\mathbf{S}$ so $\mathbf{x}_i^\top \mathbf{S}$ is monotonically nonincreasing
  Optional: $\mathbf{M}_{t+1} \leftarrow [\mathbf{M}_{t+1}]^+$ {Project onto the PSD cone}
end for
return $\mathbf{M}_T$

in Equation (7.9) for each triplet $(i, j, k) \in T$. □

Note that the large size of set T plays no role in the running time; each iteration requires $\mathcal{O}(d^2)$ work to compute the updated $\mathbf{M}$ matrix. In practice, the $\mathbf{S}$ matrix needs only be defined for all viable degrees, rather than all degrees from 1 to n, thus each matrix should take $\mathcal{O}(d \max(c))$ work to update. Assuming the node feature vectors are of bounded norm, the radius of the input data R is constant with respect to n, since each is constructed using the feature vectors of three nodes. In practice, as in the PEGASOS algorithm, we propose using $\mathbf{M}_T$ as the output instead of the average, as doing so performs better on real data, but an averaging version is easily implemented by storing a running sum of $\mathbf{M}$ matrices and dividing by T before returning.

7.4 Structure preserving metric learning

This section provides a sibling-algorithm to degree distributional metric learning, in which the graph is predicted by running a predetermined *connectivity algorithm* using a learned distance. *Structure preserving metric learning* (SPML) learns an $\mathbf{M}$ which is *structure preserving*, as defined in Definition 7.4.1 [Shaw *et al.*, To appear]. Given a connectivity algorithm $\mathcal{G}$, SPML learns a metric such that applying $\mathcal{G}$ to the input data using the learned metric produces the input adjacency matrix exactly. Possible choices for $\mathcal{G}$ include maximum weight b-matching, k-nearest neighbors, ϵ-neighborhoods, or maximum weight spanning tree.

Definition 7.4.1. *Given a graph with adjacency matrix* $\mathbf{A}$*, a distance metric parametrized by* $\mathbf{M} \in \mathbb{R}^{d \times d}$ *is* ***structure preserving*** *with respect to a connectivity algorithm* $\mathcal{G}$*, if* $\mathcal{G}(\mathbf{X}, \mathbf{M}) = \mathbf{A}$.

Algorithms for learning structure preserving metrics are similar to the algorithms described in Section 7.3. The following section describes in detail a stochastic method for learning a structure preserving embedding using the k-nearest neighbor connectivity algorithm.

7.4.1 Stochastic SPML with k-nearest neighbor

This section describes a learning algorithm that learns a structure preserving metric using the k-nearest neighbor connectivity algorithm. As in the stochastic version of DDML in Section 7.3, structure preserving embedding can be written as an objective over node-neighbor-imposter triplets.

Again, let set $T = \{(i, j, k) \mid A_{ij} = 1, A_{ik} = 0\}$ contain all triplets. Regularizing $\mathbf{M}$ with the Frobenius norm, the objective function for SPML is

$$f(\mathbf{M}) = \frac{\lambda}{2}||\mathbf{M}||_{\mathrm{F}}^2 - \frac{1}{|T|} \sum_{(i,j,k)\in T} \max(D_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) - D_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_k) + 1, 0).$$

Again using the distance transformation in Equation 7.4, and the construction of the sparse $\mathbf{C}$ matrices in Equation (7.7), the subgradient of f at $\mathbf{M}$ is the same as the partial gradient in DDML,

$$\nabla f = \lambda \mathbf{M} + \frac{1}{|T|} \sum_{(i,j,k)\in T_+} \mathbf{X}\mathbf{C}^{(i,j,k)}\mathbf{X}^\top,$$

where $T_+ = \{(i, j, k)|D_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) - D_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_k) + 1 > 0\}$. If this quantity is negative for all triplets, there exists no unconnected neighbor of a point which is closer than a point's farthest connected neighbor – precisely the structure preserving criterion for nearest neighbor algorithms. The k-nearest neighbor SPML algorithm is listed in Algorithm 7.

Stochastic SPML has the same convergence guarantees of DDML, but only learns metric parameter $\mathbf{M}$, significantly reducing the number of free variables being optimized and accordingly reducing the constant cost per iteration, leading to a much faster learning algorithm. The primary drawback is that, given a structure preserving embedding, prediction of a new graph is not obvious, since it is unknown how many neighbors to connect in a new graph. With DDML, the mapping from new nodes to target degrees is also available, so a full predictive model is possible, whereas with SPML, only a ranking of possible edges by the learned distance is available.

7.5 Experiments with DDML and SPML

We compare DDML and SPML to a variety of methods for predicting links from node features: Euclidean distances, *relational topic models* (RTM) , and traditional *support vector machines* (SVM). A simple baseline for comparison is how well the Euclidean distance metric performs at ranking possible connections. Relational topic models learn a link probability function in addition to latent topic mixtures describing each node [Chang and Blei, 2010]. For the SVM, we construct training examples consisting of the pairwise differences between node features. Training examples are labeled positive if there exists an edge between the corresponding pair of nodes, and negative if there is no edge. Because there are potentially $\mathcal{O}(n^2)$ possible examples, and the graphs are sparse, we

Algorithm 7 Structure preserving metric learning with nearest neighbor constraints and optimization with projected stochastic subgradient descent

Require: $\mathbf{A} \in \mathbb{B}^{n \times n}$, $\mathbf{X} \in \mathbb{R}^{d \times n}$, and parameters λ, T, B

$\mathbf{M}_1 \leftarrow \mathbf{I}_d$
for t from 1 to $T - 1$ **do**
  $\eta_t \leftarrow \frac{1}{\lambda t}$
  $\mathbf{C} \leftarrow \mathbf{0}_{n,n}$
  for b from 1 to B **do**
    $(i, j, k) \leftarrow$ Sample random triplet from $T = \{(i, j, k) \mid A_{ij} = 1, A_{ik} = 0\}$
    if $D_{\mathbf{M}_t}(\mathbf{x}_i, \mathbf{x}_j) - D_{\mathbf{M}_t}(\mathbf{x}_i, \mathbf{x}_k) + 1 > 0$ **then**
      $\mathbf{C}_{jj} \leftarrow \mathbf{C}_{jj} + 1$, $\mathbf{C}_{ik} \leftarrow \mathbf{C}_{ik} + 1$, $\mathbf{C}_{ki} \leftarrow \mathbf{C}_{ki} + 1$
      $\mathbf{C}_{ij} \leftarrow \mathbf{C}_{ij} - 1$, $\mathbf{C}_{ji} \leftarrow \mathbf{C}_{ji} - 1$, $\mathbf{C}_{kk} \leftarrow \mathbf{C}_{kk} - 1$
    end if
  end for
  $\nabla_t \leftarrow \mathbf{X}\mathbf{C}\mathbf{X}^\top + \lambda \mathbf{M}_t$
  $\mathbf{M}_{t+1} \leftarrow \mathbf{M}_t - \eta_t \nabla_t$
  Optional: $\mathbf{M}_{t+1} \leftarrow [\mathbf{M}_{t+1}]^+$ {Project onto the PSD cone}
end for
return $\mathbf{M}_T$

subsample the negative examples so that we include a randomly chosen equal number of negative examples as positive edges. Without subsampling, the SVM is unable to run our experiments in a reasonable time. We use the SVMPerf implementation for our SVM [Joachims, 2006], and the authors' code for RTM [Chang and Blei, 2010].

An SVM with these inputs can be interpreted as an instance of SPML using diagonal $\mathbf{M}$ and the ϵ-neighborhood connectivity algorithm, which connects points based on their distance, completely independently of the rest of the graph structure. We thus expect to see better performance using DDML and SPML in cases where the structure is important. The RTM approach is appropriate for data that consists of counts, and is a generative model which recovers a set of topics in addition to link predictions. Despite the generality of the model, RTM does not seem to perform as well as discriminative methods in our experiments, especially in the Facebook experiment where the data is quite different from bag-of-words features. For DDML and SPML, we run the stochastic algorithm with batch size 10. We skip the PSD projection step, since these experiments are only concerned with prediction, and obtaining a true metric is not immediately necessary. Both DDML and SPML are implemented in MATLAB and require only a few minutes to converge for each of the experiments below.

For DDML, the prediction task of ranking is not entirely natural to the model, since the degree component of the predictor function causes the edges to no longer have a single independent distance-based score. Two options possible for ranking from a DDML model are: (1) ignore the degree component and simply rank using the distance parameterized by the learned $\mathbf{M}$ matrix, or (2) predict using the degree component by using the degree of the training nodes, or considering the total prediction function score for a new graph created by appending each held-out edge to the observed training graph. Neither of these methods completely leverages the richness of the DDML prediction function, yet produce state-of-the-art performance.

Wikipedia articles We apply SPML to predicting links on Wikipedia pages. We first create a few subnetworks consisting of all the pages in a given category, their bag-of-words features, and their connections. We choose three categories: "graph theory topics", "philosophy concepts", and "search engines". We use a word dictionary of common words with stop-words removed. For each network, we split the data with 20% of the nodes are held out for evaluation. On the remaining

80% we cross-validate (five folds) over the parameters for each algorithm (RTM, SVM, DDML, SPML), and train a model using the best-scoring regularization parameter. For DDML and SPML, we use a sparse diagonal metric matrix $\mathbf{M}$, clamping all off-diagonal entries to zero, since the high-dimensionality of the input features reduces the benefit of cross-feature weights. On the held-out nodes, we task each algorithm to rank the unknown edges according to distance (or another measure of link likelihood), and compare the accuracy of the rankings using *receiver operator characteristic* (ROC) curves. Table 7.1 lists the statistics of each category and the average area under the curve (AUC) over three train/test splits for each algorithm. ROC curves for the "philosophy concepts" category is shown in Figure 7.3. For the small Wikipedia categories, SPML and DDML provide a distinct advantage over other methods. One possible explanation for why the SVM is unable to gain performance over Euclidean distance is that the wide range of degrees for nodes in these graphs makes it difficult to find a single threshold that separates edges from non-edges. In particular, the "search engines" category had an extremely skewed degree distribution, and is where SPML shows the greatest improvement.

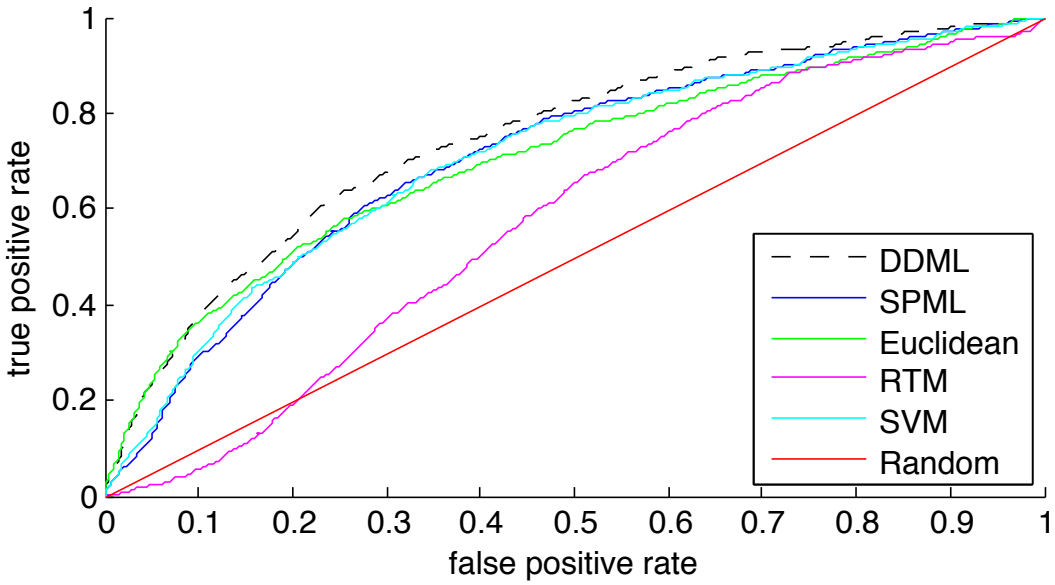


Figure 7.3: ROC curve for various algorithms on the "philosophy concepts" category.

We also apply SPML and DDML to a larger subset of the Wikipedia network, by collecting

word counts and connections of 100,000 articles in a breadth-first search rooted at the article "Philosophy". The experimental setup is the same as previous experiments, but we use a 0.5% sample of the nodes for testing. We stop each stochastic optimization after ten minutes on a desktop computer. The resulting AUC on the edges of the held-out nodes is listed in Table 7.1 as the "Philosophy Crawl" dataset. The SVM and RTM do not scale to data of this size, whereas SPML offers a clear advantage over using Euclidean distance for predicting links, while DDML also provides a lift.

Table 7.1: Wikipedia (top), Facebook (bottom) dataset and experiment information. Shown below: number of nodes n, number of edges $|E|$, dimensionality d, and AUC performance. The small Wikipedia categories (*) are ranked by DDML using degree information from the training graph, and the other data sets are ranked using only the learned distance.

	n	$\|E\|$	d	Euclid.	RTM	SVM	SPML	DDML
Graph Theory	223	917	6695	0.624	0.591	0.610	**0.722**	0.691*
Philosophy Concepts	303	921	6695	0.705	0.571	0.708	0.707	**0.746***
Search Engines	269	332	6695	0.662	0.487	0.611	**0.742**	0.725*
Philosophy Crawl	100k	4m	7702	0.547	–	–	**0.601**	0.562
Harvard	1937	48k	193	0.764	0.562	0.839	**0.854**	0.848
MIT	2128	95k	173	0.702	0.494	0.784	**0.801**	0.797
Stanford	3014	147k	270	0.718	0.532	0.784	0.808	**0.810**
Columbia	3050	118k	251	0.717	0.519	0.796	0.818	**0.821**

Facebook social networks Applying DDML and SPML to social network data allows us to more accurately predict who will become friends based on the profile information for those users. We use Facebook data [Traud *et al.*, 2011], where we have a small subset of anonymized profile information for each student of a university, as well as friendship information. The profile information consists of gender, status (meaning student, staff, or faculty), dorm, major, and class year. Similarly to the Wikipedia experiments in the previous section, we compare DDML and SPML to Euclidean, RTM, and SVM. For DDML and SPML, we learn a full $\mathbf{M}$, including cross-feature weights. For

each person, we construct a sparse feature vector where there is one feature corresponding to every possible dorm, major, etc. for each feature type. We select only people who have indicated all five feature types on their profiles. Table 7.1 shows details of the Facebook networks for the four schools we consider: Harvard, MIT, Stanford, and Columbia. We perform a separate experiment for each school, holding out 20% of the nodes for testing. We use the training data to select parameters via five-fold cross validation, and train a model. The AUC performance on the held-out edges are also listed in Table 7.1. It is clear from the quantitative results that structural information is contributing to higher performance for DDML and SPML as compared to other methods.

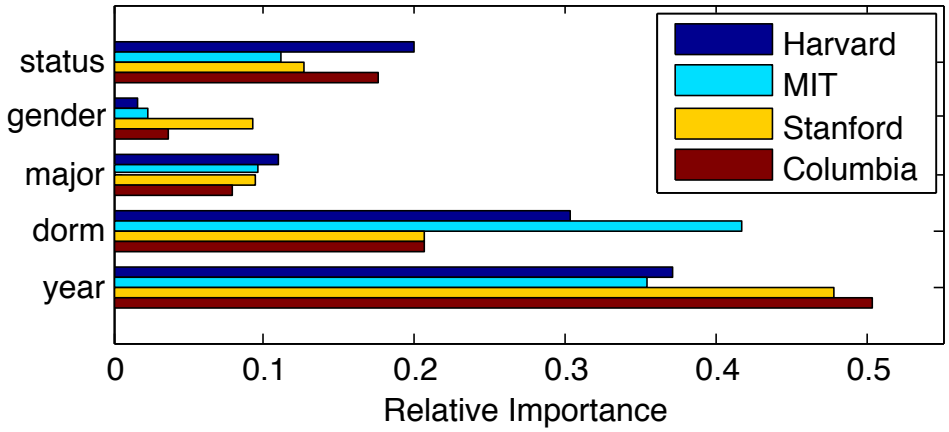


Figure 7.4: Comparison of Facebook social networks from four schools in terms of feature importance computed from the learned structure preserving metric.

By looking at the weight of the diagonal values in $\mathbf{M}$ learned by SPML, normalized by the total weight, we can determine which feature differences are most important for determining connectivity. Figure 7.4 shows the normalized weights averaged by feature types for Facebook data. Here we see the feature types compared across four schools. For all schools except MIT, the graduating year is most important for determining distance between people. For MIT, dorms are the most important features. A possible explanation for this difference is that MIT is the only school in the list that makes it easy for students to stay in a residence for all four years of their undergraduate program, and therefore which dorm one lives in may affect more strongly the people they connect to.

7.6 Discussion of learning from and predicting networks

This chapter proposes techniques for solving the problem of learning the parameters to a degree-aware distribution over graphs for the purpose of network prediction. Initial motivation is provided by a synthetic experiment where knowing the degrees of a network significantly benefits prediction from observed data. Two variants of metric learning algorithms are then proposed for learning a mapping from node features to edge potentials, as well as degree potentials in degree distributional metric learning. These algorithms are initial attempts to leverage the rich space of models representable in the degree-based subgraph estimation framework.

Another novel aspect of the techniques in this chapter is the merging of structural connectivity information with nonstructural node data. There is a surprising scarcity in the literature of methods that simultaneously consider both aspects. Many link prediction algorithms, as reviewed in Chapter 2, model only the links of graph data. When learning a structure preserving metric, the learned metric attempts to correlate the node features with the structural behavior as much as possible, while learning a degree distributional metric simultaneously learns a structural component while also correlating the features to structure.

Chapter 8

Conclusions and Discussion

As much novel content that fills the chapters preceding this final chapter, this thesis is but an early step in its important research direction. This thesis provides methods for degree-based graph prediction and some extensions and variations, focusing on the inclusion of the structural measure of degree in computationally efficient and exact structure estimation.

8.1 Contributions

This thesis contributes the following original concepts to the current state-of-the-art:

- I describe a probability distribution over graph structures that is dependent on the existence of edges as well as the degrees of each node, and derive a procedure for efficient maximum likelihood inference of graph structure. The procedure reduces the maximum likelihood estimation to the well-studied maximum weight b-matching problem.
- I derive a distributable message-passing algorithm from belief propagation for solving maximum weight b-matchings, which is proven to converge to the true solution under well-defined conditions. This is one of few known convergence proofs for loopy max-product belief propagation.
- I further derive various scalability improvements for the belief propagation algorithm, allowing the exact solution using belief propagation of much larger dense b-matching problems than previously possible.

- I provide algorithms exploiting degree-based arguments to solve the collaborative filtering problem, a variant of which achieves state-of-the-art performance with a convex optimization using counting statistics interpretable as degrees in a multi-relational graph.
- I derive a learning algorithm for learning distribution parameters for degree-based graph estimation by implementing maximum margin structured prediction. A stochastic variant of this algorithm allows learning in large-scale settings.

8.2 Open questions and future work

Learning degree information from observed networks reveals a significant challenge for structure-based network analysis. In practically all observed networks, the available data is incomplete. For example, any measured human social network is incomplete until all humans, real or imaginary, who have ever existed are included in the data. Only with the complete set of all possible human social relationships can we claim to have observed the true structure. Concerning degrees, an approach to handle this incompleteness is to assume that all observed degrees bound the true degrees from below, but how to avoid overestimating true, latent degrees remains an open problem. With more complex measures of structure, the incompleteness of measured networks is too an important open problem.

Relatedly, the performance evaluation of network prediction is typically done, as in the experiments within this thesis, by measures derived for independent predictions, such as accuracy, ranking, precision, or recall. These measures are ignorant of structure, and thus it is possible that a prediction that performs well in these independent measures may in fact be unrealistic structurally. Theoretical and empirical exploration of structure-aware measures of network prediction quality will be a useful direction for future research.

The relationship between the belief propagation algorithm described in Chapter 4 and the linear programming (LP) relaxation of b-matching added insight into the relationship between max-product and LP relaxations in general. Various work has been published since exploring message-passing algorithms based directly on solving LP relaxations of the general inference problem. Progress continues in the research community on understanding the properties of these message-passing inference algorithms.

Better understanding and practical implementation of a fully distributed version of these message-passing algorithms, especially the b-matching solver in this thesis will be of great interest for network research. Since the algorithm easily distributes between nodes in a graph, performing network prediction by having nodes pass messages to their neighbors allows for graph prediction to be computed by the graph itself, possibly allowing privacy-preserving properties. Furthermore, recent advances in belief propagation analysis has provided an example of fully-distributed solution detection for the min-cost flow problem [Gamarnik *et al.*, 2010], which generalizes b-matching. Thus, formulating the distributed stopping criterion specifically for b-matching will be a useful addition to the established algorithms.

Finally, Chapter 7 deals with the combination of node data with network connectivity. All real networks have node attributes, yet they are often not measured or intentionally omitted because there is a lack of established methods to elegantly handle both network connectivity and node data. This is unfortunate and embarrassing, and new algorithms in addition to the ones proposed in this thesis for simultaneously modeling node attributes and network connectivity will have immediate impact for network researchers.

8.3 Discussion

The ideas in this thesis represent a significant portion of the research I have done during my doctoral program, which is a relatively arbitrary window in the continuum of research by the machine learning, statistics, artificial intelligence, and data mining communities as a whole. While most of this thesis text is aimed to construct a cohesive narrative with a figurative beginning, middle, and end, this final paragraph aims to do the exact opposite; the work described here builds off that of my contemporaries around the world, and will lead to more work by others and me that will aim to immediately make the techniques in this thesis obsolete. There is no beginning or end, only a middle. The ideas in this thesis are attempts at solving extraordinarily challenging problems in machine learning with network data, and, while some problems have been resolved by the work here, countless problems remain. Even within the smaller subarea of learning using degree information, significant challenges are still unsolved, and the full potential of using degree information remains a future goal.

Part III

Appendices

Appendix A

Miscellaneous Extensions

A.1 Approximating the matrix permanent with matching belief propagation

The permanent is a scalar quantity computed from a matrix and has been an active topic of research for well over a century. It plays a role in cryptography and statistical physics where it is fundamental to Ising and dimer models. While the determinant of an $n \times n$ matrix can be evaluated exactly in sub-cubic time, efficient methods for computing the permanent have remained elusive. Since the permanent is #P-complete, efficient exact evaluations cannot be found in general. The best exact methods improve over brute force ($\mathcal{O}(n!)$) and include Ryser's algorithm [Ryser, 1963; Servedio and Wan, 2005] which requires as many as $\Theta(n2^n)$ arithmetic operations. Recently, promising fully-polynomial randomized approximate schemes (FPRAS) have emerged which provide arbitrarily close approximations. Many of these methods build on initial results by Broder [Dagum and Luby, 1992] who applied Markov chain Monte Carlo (a popular tool in machine learning and statistics) for sampling perfect matchings to approximate the permanent. Significant progress has produced an FPRAS that can handle arbitrary $n \times n$ matrices with non-negative entries [Jerrum *et al.*, 2004]. The method uses Markov chain Monte Carlo and only requires a polynomial order of samples.

However, while these methods have tight theoretical guarantees, they carry expensive constant factors, not to mention relatively high polynomial running times that discourage their usage in practical applications. In particular, we have experienced that the prominent algorithm in [Jerrum

et al., 2004] is slower than Ryser's exact algorithm for any feasible matrix size, and project that it only becomes faster around $n > 30$.

It remains to be seen if other approximate inference methods can be brought to bear on the permanent. For instance, loopy belief propagation has also recently gained prominence in the machine learning community. The method is exact for singly-connected networks such as trees. In certain special loopy graph cases, including graphs with a single loop, bipartite matching graphs [Bayati *et al.*, 2005] and bipartite multi-matching graphs [Huang and Jebara, 2007], the convergence of BP has been proven. In more general loopy graphs, loopy BP still maintains some surprising empirical success. Theoretical understanding of the convergence of loopy BP has recently been improved by noting certain general conditions for its fixed points and relating them to minima of Bethe free energy. This chapter proposes belief propagation for computing the permanent and investigates some theoretical and experimental properties [Huang and Jebara, 2009b].

A.1.1 The permanent as a partition function

Given an $n \times n$ non-negative matrix W, the matrix permanent is

$$\sum_{\pi \in S_n} \prod_{i=1}^{n} W_{i\pi(i)}. \tag{A.1}$$

Here S_n refers to the symmetric group on the set $\{1, \ldots, n\}$, and can be thought of as the set of all permutations of the columns of W. To gain some intuition toward the upcoming analysis, we can think of the matrix W as defining some function $f(\pi; W)$ over S_n. In particular, the permanent can be rewritten as

$$\text{per}(W) = \sum_{\pi \in S_n} f(\pi; W),$$
$$\text{where} \qquad f(\pi; W) = \prod_{i=1}^{n} W_{i\pi(i)}.$$

The output of f is non-negative, so we consider f a density function over the space of all permutations.

If we think of a permutation as a perfect matching or assignment between a set of n objects A and another set of n object B, we relax the domain by considering all possible assignments of imperfect matchings for each item in the sets.

Consider the set of assignment variables $X = \{x_1, \ldots, x_n\}$, and the set of assignment variables $Y = \{y_1, \ldots, y_n\}$, such that $x_i, y_j \in \{1, \ldots, n\}, \forall i, j$. The value of the variable x_i is the assignment for the i'th object in A, in other words the value of x_i is the object in B being selected (and vice versa for the variables y_j).

$$\begin{aligned} \phi(x_i) &= \sqrt{W_{ix_i}}, \quad \phi(y_j) = \sqrt{W_{y_j j}}, \\ \psi(x_i, y_j) &= I(\neg(j = x_i \oplus i = y_j)). \end{aligned}$$

We square-root the matrix entries because the factor formula multiplies both potentials for the x and y variables. We use $I()$ to refer to an indicator function such that $I(\text{true}) = 1$ and $I(\text{false}) = 0$. Then the ψ function outputs zero whenever any pair (x_i, y_j) have settings that cannot come from a true permutation (a perfect matching). Specifically, if the i'th object in A is assigned to the j'th object in B, the j'th object in B must be assigned to the i'th object in A (and vice versa) or else the density function goes to zero. Given these definitions, we can define the equivalent density function that subsumes $f(\pi)$ as follows:

$$\hat{f}(X, Y) = \prod_{i,j} \psi(x_i, y_j) \prod_k \phi(x_k)\phi(y_k).$$

This permits us to write the following equivalent formulation of the permanent: $\text{per}(W) = \sum_{X,Y} f(X, Y)$. Finally, if we convert density function $\hat{f}$ into a valid probability, simply add a normalization constant to it, producing:

$$p(X, Y) = \frac{1}{Z(W)} \prod_{i,j} \psi(x_i, y_j) \prod_k \phi(x_k)\phi(y_k). \tag{A.2}$$

The normalizer or partition function $Z(W)$ is the sum of $f(X, Y)$ for all possible inputs X, Y. Therefore, the partition function of this distribution is the matrix permanent of W.

Bethe Free Energy To approximate the partition function, we use the Bethe free energy approximation. The Bethe free energy of our distribution given a belief state b is

$$\begin{aligned} F_{\text{Bethe}} &= -\sum_{ij}\sum_{x_i,y_j} b(x_i,y_j)\ln\psi(x_i,y_j)\phi(x_i)\phi(y_j) \\ &\quad +\sum_{ij}\sum_{x_i,y_j} b(x_i,y_j)\ln b(x_i,y_j) \\ &\quad -(n-1)\sum_{i}\sum_{x_i} b(x_i)\ln b(x_i) \\ &\quad -(n-1)\sum_{j}\sum_{y_j} b(y_j)\ln b(y_j) \end{aligned} \tag{A.3}$$

The belief state b is a set of pseudo-marginals that are only locally consistent with each other, but need not necessarily achieve global consistency and do not have to be true marginals of a single global distribution. Thus, unlike the distributions evaluated by the exact Gibbs free energy, the Bethe free energy involves pseudo-marginals that do not necessarily agree with a true joint distribution over the whole state-space. The only constraints pseudo-marginals of our bipartite distribution obey (in addition to non-negativity) are the linear local constraints:

$$\begin{aligned} \sum_{y_j} b(x_i,y_j) &= b(x_i), \quad \sum_{x_i} b(x_i,y_j) = b(y_j),\ \forall i,j, \\ \sum_{x_i,y_j} b(x_i,y_j) &= 1. \end{aligned}$$

The class of true marginals is a subset of the class of pseudo-marginals. In particular, true marginals also obey the constraint $\sum_{X\setminus x} p(X) = p(x)$, which pseudo-marginals are free to violate.

We will use the approximation

$$\text{per}(W) \approx \exp\left(-\min_b F_{\text{Bethe}}(b)\right) \tag{A.4}$$

Belief propagation The canonical algorithm for (locally) minimizing the Bethe free energy is *belief propagation*. We use the dampened belief propagation described in [Heskes, 2003], which the author derives as a (not necessarily convex) minimization of Bethe free energy. Belief propagation is a message passing algorithm that iteratively updates messages between variables that define the local beliefs. Let $m_{x_i}(y_j)$ be the message from x_i to y_j. Then the beliefs are defined by the messages as follows:

$$b^t(x_i,y_j) \propto \psi(x_i,y_j)\phi(x_i)\phi(y_j)\prod_{k\neq j} m^t_{y_k}(x_i)\prod_{\ell\neq i} m^t_{x_\ell}(y_j)$$

$$b^t(x_i) \propto \phi(x_i) \prod_k m^t_{y_k}(x_i), \quad b^t(y_j) \propto \phi(y_j) \prod_k m^t_{x_k}(y_j) \tag{A.5}$$

In each iteration, the messages are updated according to the following update formula:

$$m^{\text{new}}_{x_i}(y_j) = \sum_{x_i} \left[\phi(x_i)\psi(x_i, y_j) \prod_{k \neq j} m^t_{y_k}(x_i) \right] \tag{A.6}$$

Finally, we dampen the messages to encourage a smoother optimization in log-space.

$$\ln m^{t+1}_{x_i}(y_j) \leftarrow \ln m^t_{x_i}(y_j) + \epsilon \left[\ln m^{\text{new}}_{x_i}(y_j) - \ln m^t_{x_i}(y_j) \right] \tag{A.7}$$

We use ϵ as a dampening rate as in [Heskes, 2003] and dampen in log space because the messages of BP are exponentiated Lagrange multipliers of Bethe optimization [Heskes, 2003; Yedidia *et al.*, 2005; Yuille, 2002]. We next derive faster updates of the messages (A.6) and the Bethe free energy (A.3) with some careful algebraic tricks.

Algorithmic Speedups Computing sum-product belief propagation quickly for our distribution is challenging since any one variable sends a message vector of length n to each of its n neighbors, so there are $2n^3$ values to update each iteration. One way to ease the computational load is to avoid redundant computation. In Equation (A.6), the only factor affected by the value of y_j is the ψ function. Therefore, we can explicitly define the update rules based on the ψ function, which will allow us to take advantage of the fact that the computation for each setting of y_j is similar. When $y_j \neq i$, we have

$$\begin{aligned} m^{\text{off}}_{x_i y_j} &= \left(\sum_{x_i \neq j} \phi(x_i) \prod_{k \neq j} m_{y_k}(x_i) \right) \\ &= \left(\sum_{x_i \neq j} \phi(x_i) m^{\text{on}}_{y_{x_i} x_i} \prod_{k \neq j, k \neq x_i} m^{\text{off}}_{y_k x_i} \right). \end{aligned} \tag{A.8}$$

When $y_j = i$,

$$\begin{aligned} m^{\text{on}}_{x_i y_j} &= \left(\phi(x_i = j) \prod_{k \neq j} m_{y_k}(x_i = j) \right) \\ &= \left(\phi(x_i = j) \prod_{k \neq j} m^{\text{off}}_{y_k x_i} \right). \end{aligned} \tag{A.9}$$

We have reduced the full message vectors to only two possible values: m^{off} is the message for when the variables are not matched and m^{on} is for when they are matched. We further simplify the messages by dividing both values by $m^{\text{off}}_{x_i y_j}$. This gives us

$$\begin{aligned} m^{\text{off}}_{x_i y_j} &= 1 \\ m^{\text{on}}_{x_i y_j} &= \frac{\phi(x_i = j) \prod_{k \neq j} m^{\text{off}}_{y_k x_i}}{\sum_{x_i \neq j} \phi(x_i) m^{\text{on}}_{y_{x_i} x_i} \prod_{k \neq j, k \neq x_i} m^{\text{off}}_{y_k x_i}} \\ &= \frac{\phi(x_i = j)}{\sum_{k \neq j} \phi(x_i = k) m^{\text{on}}_{y_k x_i}} \end{aligned} \tag{A.10}$$

We can now define a fast message update rule that only needs to update one value between each variable.

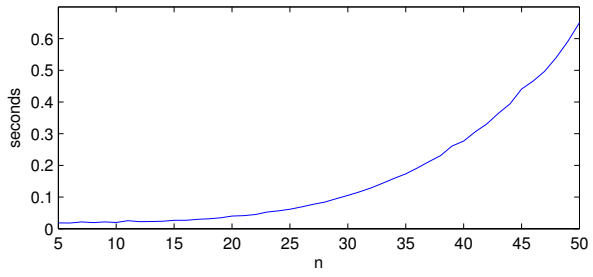
$$m^t_{x_i y_j} \leftarrow \frac{1}{Z} \phi(x_i = j) / \sum_{k \neq j} \phi(x_i = k) m^t_{y_k x_i} \tag{A.11}$$

We can rewrite the belief update formulas using these new messages.

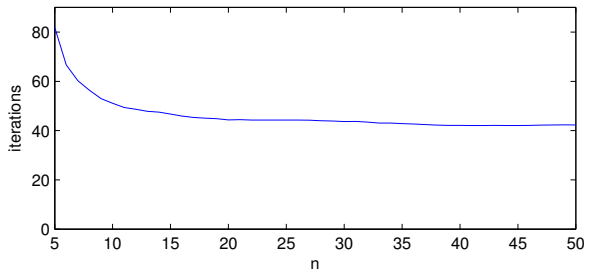
$$\begin{aligned} b(x_i = j, y_j = i) &\propto \phi(x_i)\phi(y_j) \\ b(x_i \neq j, y_j \neq i) &\propto \phi(x_i)\phi(y_j) m_{y_{x_i} x_i} m_{x_{y_j} y_j} \\ b(x_i) &\propto \phi(x_i) m_{y_{x_i} x_i}, \\ b(y_j) &\propto \phi(y_j) m_{x_{y_j} y_j} \end{aligned} \tag{A.12}$$

With the simplified message updates, each iteration takes $\mathcal{O}(n)$ operations per node, resulting in an algorithm that takes $\mathcal{O}(n^2)$ operations per iteration. We demonstrate experimentally that the algorithm converges to within a certain tolerance in a constant number of iterations with respect to n, so in practice the $\mathcal{O}(n^3)$ operations it takes to compute Bethe free energy is the asymptotic bottleneck of our algorithm.

Convergence One important open question about this work is whether or not we can guarantee convergence. Empirically, by initializing belief propagation to various random points in the feasible space, we found BP still converged to the same solution. The max-product algorithm is guaranteed to converge to the correct maximum matching [Bayati *et al.*, 2005; Huang and Jebara, 2007] via arguments on the unwrapped computation tree of belief propagation. The matching graphical model does not not meet the sufficient conditions provided in [Heskes, 2006] nor does our distribution fit



(a) Running time



(b) Iterations

Figure A.1: (a) Average running time until convergence of BP for $5 \leq n \leq 50$. (b) Number of iterations.

the criteria for non-convex convergence provided in [Tatikonda and Jordan, 2002] and [Heskes, 2004].

The Bethe approximation of the permanent has been proven to be a minimizer of a convex function, which makes the true matrix permanent lower bounded by the Bethe approximation [Vontobel, 2010], as is empirically observed in the experiments that follow.

A.1.2 Empirical evaluation of Bethe permanent approximation

In this section we evaluate the performance of this algorithm in terms of running time and accuracy, and finally we exemplify a possible usage of the approximate permanent as a kernel function.

Running Time We run belief propagation to approximate the permanents of random matrices of sizes $n = [5, 50]$, recording the total running time and the number of iterations to convergence. Surprisingly, the number of iterations to convergence initially *decreases* as n grows, but appears to

remain constant beyond $n > 10$ or so. Running time still increases because the cost of updating each iteration well subsumes the decrease in iterations to convergence.

In our implementation, we check for convergence by computing the absolute change in all the messages from the previous iteration, and consider the algorithm converged if the sum of all the changes of all n^3 messages is less than $1e-10$. In all cases, the resulting beliefs are consistent with each other within comparable precision to our convergence threshold. These experiments were run on a a 2.4 Ghz Intel Core 2 Duo Apple Macintosh running Mac OS X 10.5. The code is in C and compiled using `gcc` version 4.0.1. Plots of measured running times and iterations are in Figure A.1.

Table A.1: Normalized Kendall distances between the rankings of random matrices based on their true permanents and the rankings based on approximate permanents. See Figure A.2 for plots of the approximations.

n	Bethe	Sampling	Det.	Diag.
10	**0.00023**	0.0248	0.3340	0.0724
8	**0.0028**	0.1285	0.4995	0.4057
5	**0.0115**	0.0914	0.4941	0.3834

Accuracy of Approximation We evaluate the accuracy of our algorithm by creating 1000 random matrices of sizes 5, 8 and 200 matrices of size 10. The entries of each of these matrices were randomly drawn from a uniform distribution in the interval $[0, 50]$. We computed the true permanents of these matrices, then computed approximate permanents using our Bethe approximation. We also computed an approximate using a naive sampling method, where we sample by choosing random permutations and storing a cumulative sum of each permutation's corresponding product. We sampled for the same amount of actual time our belief propagation algorithm took to converge. Finally we also computed two weak approximations: the determinant and the scaled product of the diagonal entries.

In order to be able to compare to the true permanent, we had to limit this analysis to small matrices. However, since MCMC sampling methods such as in [Jerrum *et al.*, 2004] take $\mathcal{O}(n^{10})$ time to reach less than some ϵ error, as matrix size increases, the precision achievable in comparable time to our algorithm would decrease. We scale the cumulative sum by $\frac{n!}{s}$, where s is the number

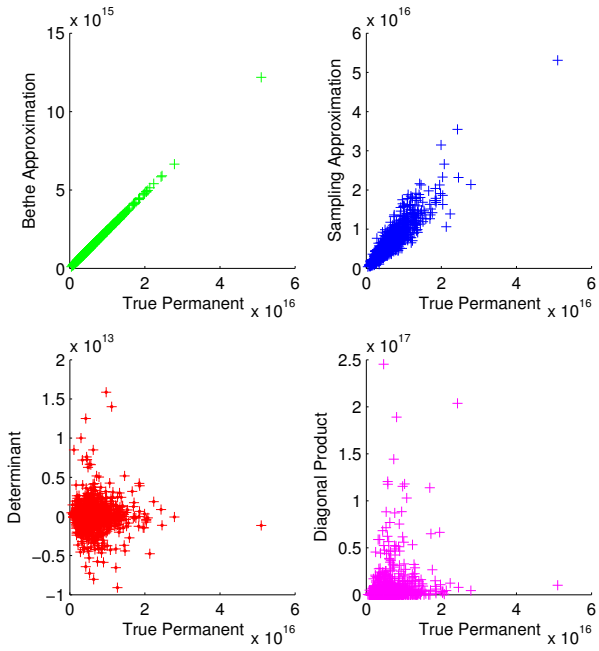


Figure A.2: Plots of the approximated permanent versus the true permanent using four different methods. It is important to note that the scale of the y-axis varies from plot to plot. The diagonal is extremely erratic and the determinant underestimates so much that it is barely visible on the log scale. Sampling approximates values much closer in absolute distance to the true permanent but does not provide monotonicity in its approximations. Typically, this is more important than absolute accuracy. Here we illustrate the results from the $n = 8$ case. We report results for $n = 5$ and 10 in Table A.1.

of samples. This is the ratio of the total possible permutations and the number of samples.

In our experiments, determinants and the products of diagonals are neither accurate nor consistent approximations of the permanent. Sampling, however, is accurate with respect to absolute distance to the permanent, so for applications where that is most important, it may be best to apply some sort of sampling method. Our Bethe approximation seems the most consistent. While the approximations of the permanent are off by a large amount, they seem to be consistently off by some monotonic function of the true permanent. In many cases, this virtue is more important than the absolute accuracy, since most applications requiring a matrix permanent likely compare the

permanents of various matrices. These results are visualized for $n = 8$ in Figure A.2.

To measure the monotonicity and consistency of these approximations, we consider the Kendall distance [Fagin *et al.*, 2003] between the ranking of the random matrices according to the true permanent and their rankings according to the approximations. Kendall distance is a popular way of measuring the distance between two permutations. The Kendall distance between two permutations π_1 and π_2 is

$$D_{\text{Kendall}}(\pi_1, \pi_2) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} I\left((\pi_1(i) < \pi_1(j)) \wedge (\pi_2(i) > \pi_2(j))\right).$$

In other words, it is the total number of pairs where π_1 and π_2 disagree on the ordering. We normalize the Kendall distance by dividing by $\frac{n(n-1)}{2}$, the maximum possible distance between permutations, so the distance will always be in the range $[0, 1]$. Table A.1 lists the Kendall distances between the true permanent ranking and the four approximations. The Kendall distance of the Bethe approximation is consistently less than that of our sampler.

Approximate Permanent Kernel To illustrate a possible usage of an efficient permanent approximation, we use a recent result [Cuturi, 2007] proving that the permanent of a valid kernel matrix between two sets of points is also a valid kernel between point sets. Since the permanent is invariant to permutation, we decided to try a few classification tasks using an approximate permanent kernel. The permanent kernel is computed by first computing a valid subkernel between a pairs of elements in two sets, then the permanent of those subkernel evaluations is taken as the kernel value between the data. Surprisingly, in experiments the kernel matrix produced by our algorithm was a valid positive definite matrix. This discovery opens up some intriguing questions to be explored later.

We ran a similar experiment to the tests used to evaluate *permutation invariant SVMs* [Shivaswamy and Jebara, 2006], where we took a the first 200 examples of each of the Cleveland Heart Disease, Pima Diabetes, and Ionosphere datasets from the UCI repository [Asuncion and Newman, 2007], and randomly permuted the features of each example, then compare the result of training an SVM on this shuffled data. We also provide the performance of the kernels on the unshuffled data for comparison. After normalizing the features of the data to the $[0, 1]^D$ box, we chose three reasonable settings of σ for the RBF kernels and cross validated over various settings of the regularization parameter C. We used RBF kernels between pairs of data as the permanent subkernel. Finally, we

Table A.2: Left: Error rates of running SVM using various kernels on the original three UCI datasets and data where the features are shuffled randomly for each datum. Right: UCI resampled pendigits data with order of points removed. Error rates of 1-versus-all multi-class SVM using various kernels.

Kernel	**Heart**	**Pima**	**Ion.**
Original Linear	0.1600	0.2600	0.1640
Orig. RBF $\sigma = 0.3$	0.2908	0.3160	0.1240
Orig. RBF $\sigma = 0.5$	0.2158	0.3220	0.0760
Orig. RBF $\sigma = 0.7$	0.1912	0.2760	0.0960
Shuffled Linear	0.2456	0.3080	0.2640
Shuff. RBF $\sigma = 0.3$	0.4742	0.3620	0.4840
Shuff. RBF $\sigma = 0.5$	0.3294	0.3140	0.3580
Shuff. RBF $\sigma = 0.7$	0.2964	0.3280	0.2700
Bethe $\sigma = 0.3$	0.2192	**0.2900**	**0.1000**
Bethe $\sigma = 0.5$	**0.2140**	**0.2900**	0.1380
Bethe $\sigma = 0.7$	0.2164	0.2920	0.1380

Kernel	**PenDigits**
Sorted Linear	0.3960
Sorted RBF $\sigma = 0.2$	0.4223
Sorted RBF $\sigma = 0.3$	0.3407
Sorted RBF $\sigma = 0.5$	0.3277
Shuffled Linear	0.7987
Shuff. RBF $\sigma = 0.2$	0.9183
Shuff. RBF $\sigma = 0.3$	0.9120
Shuff. RBF $\sigma = 0.5$	0.8657
Bethe $\sigma = 0.2$	0.1463
Bethe $\sigma = 0.3$	**0.1190**
Bethe $\sigma = 0.5$	0.1707

report the average error over 50 random splits of 150 training points and 50 testing points. Not surprisingly, the permanent kernel is robust to the shuffling and outperforms the standard kernels.

We also tested the Bethe kernel on the pendigits dataset, also from the UCI repository. The original pendigits data consists of stylus coordinates of test subjects writing digits. We used the preprocessed version that has been resampled spatially and temporally. However, we omit the order information and treat the input as a cloud of unordered points. Since there is a natural spatial interpretation of this data, so we compare to sorting by distance from origin, a simple method of handling unordered data. We chose slightly different σ values for the RBF kernels. For this dataset, there are 10 classes, one for each digit, so we used a one-versus-all strategy for multi-class classification. Here we averaged over only 10 random splits of 300 training points and 300 testing points (see Table A.2).

Based on our experiments, the permanent kernel typically does not outperform standard kernels when permutation invariance is not inherently necessary in the data, but when we induce the necessity of such invariance, its utility becomes clear.

A.1.3 Discussion and future directions

We have described an algorithm based on BP over a specific distribution that allows an efficient approximation of the $\#P$ matrix permanent operation. We write a probability distribution over matchings and use Bethe free energy to approximate the partition function of this distribution. The algorithm is significantly faster than sampling methods, but attempts to minimize a function that approximates the permanent. Therefore it is limited by the quality of the Bethe approximation so it cannot be run longer to obtain a better approximation like sampling methods can. However, we have shown that even on small matrices where sampling methods can achieve extremely high accuracy of approximation, our method is better behaved than sampling, which can approach the exact value from above or below.

In the future, we can try other methods of approximating the partition function such as generalized belief propagation [Yedidia *et al.*, 2005], which takes advantage of higher order Kikuchi approximations of free energy. Unfortunately the structure of our graphical model causes higher order interactions to become expensive quickly, since each variable has exactly N neighbors. Similarly, the bounds on the partition function in [Wainwright *et al.*, 2002] are based on spanning subtrees

in the graph, and again the fully connected bipartite structure makes it difficult to capture the true behavior of the distribution with trees.

Finally, the positive definiteness of the kernels we computed is surprising, and requires further analysis. The exact permanent of a valid kernel forms a valid Mercer kernel [Cuturi, 2007] because it is a sum of positive products, but since our algorithm outputs the results of an iterative approximation of the permanent, it is not obvious why the resulting output would obey the positive definite constraints.

A.2 Heuristic extensions for b-matching belief propagation

While belief propagation is guaranteed to converge under certain conditions, a practical implementation of belief propagation for b-matching benefits from some heuristics. In particular, ideas in this section focus on heuristic approaches to handle the inputs that do not fit the convergence criteria. First, we describe a procedure to detect oscillating messages, allowing belief propagation to exit early if it will never converge. Second, we describe heuristics to handle these outputs of unfinished belief propagation, which are not feasible solutions for the perfect b-matching problem.

A.2.1 Oscillation detection

Since belief propagation for b-matching, or for any problem, may not converge, a mechanism for detecting oscillation in the messages useful tool for real implementations of belief propagation on loopy graphs. In practice, iterative algorithms such as belief propagation are implemented with a maximum iteration count, which may be quite large. When beliefs oscillate, the algorithm will certainly run until the maximum iteration is reached, which may waste significant computation if the oscillation can instead be detected earlier.

Many cases of oscillating belief propagation observed empirically involve repeated nearly exact states. Since the belief update is deterministic, any repeated state directly implies an infinite loop of oscillation. For these cases, a we engineer and implement a simple data structure that detects identical states (within some tolerance).

For each system-wide belief state $\{\alpha, \beta\}$, we compute an integer checksum, and store these checksum values in a set data structure (*e.g.*, a hash table). At each subsequent iteration, we check

if we have previously had a belief state with the same checksum. If we have not, the current state is certainly not a previous state and belief propagation if not oscillating. If however the checksum has been seen previously, the current state may be a previously seen state, but not necessarily due to hash function collisions. When a collision occurs at this level, we escalate to a secondary set that stores the exact state with a different hash function. Since the secondary set stores the full belief state, we avoid false-positives at the secondary level. The oscillation detector algorithm is listed in Algorithm 8.

Algorithm 8 Oscillation detector for belief propagation.

Require: State $\{\alpha, \beta\}$, hash function $h(\{\alpha, \beta\}) \mapsto \mathbb{Z}$, primary set data structure H_1 storing integers, and secondary set H_2 storing belief states

$x \leftarrow h(\{\alpha, \beta\})$
if x is in H_1 **then**
  if $\{\alpha, \beta\}$ is in H_2 **then**
    return True {Belief propagation is oscillating.}
  else
    Insert $\{\alpha, \beta\}$ into H_2
  end if
else
  Insert x into H_1
end if
return False

Since we expect a significantly larger set of unique states than repeated states, the two-tier set structure allows for efficient storage of unique state checksums, each of which is a single $\mathcal{O}(1)$ integer per iteration, and, only when a repeated state is possible, a full $\mathcal{O}(n)$ belief state is stored. Thus, with a good primary hash function, for the $\mathcal{O}(n)$ iterations necessary for convergence, this oscillation detection requires $\mathcal{O}(n)$ total additional storage.

If the primary hash function produces too many collisions of hash value, the secondary set may store too many full belief states, requiring $\mathcal{O}(n)$ storage per iteration. Since this should not happen except when the hash function fails, we simply limit the secondary set to a fixed size and randomly replace previous states. Given enough repeated states, such a setup eventually detects oscillation,

albeit less immediately than if the hash function properly distinguishes unique states.

We implement the oscillation detector with hash tables using *quadratic probing*, increasing the underlying array size and rehashing as needed [Cormen *et al.*, 2001]. The hash functions multiply the belief vectors by 1×10^6, and sum the rounded integer values of these. A more clever hash function could produce fewer collisions, but we find in practice, this simple hash function produces sufficiently few collisions. In the secondary hash table, we consider belief states equal if their vector representations have ℓ_1 distance of less than 1×10^{-16}, to account for small floating point inaccuracies. Computing each checksum takes $\mathcal{O}(n)$ work at the beginning of each iteration, and performing the set operations requires $\mathcal{O}(1)$ at the primary level and $\mathcal{O}(n)$ at the secondary level, so the oscillation detector adds some minimal extra overhead to belief propagation.

A.2.2 Heuristic to find feasible b-matching without convergence

While finding a feasible b-matching is itself a combinatorial optimization in arbitrary graphs, in many settings, especially in machine learning applications, the underlying graph is dense, or fully-connected, in which case constructing a feasible b-matching is significantly easier. This section provides a heuristic method for completing a b-matching when belief propagation fails to converge.

Empirical evidence indicates that, for many realistic inputs, belief propagation will converge to valid b-matching states for most of any graph, while a small group of nodes oscillate. Since all edges are possible in fully connected settings, we consider the quality of the solution generated by arbitrarily correcting the oscillating nodes.

Since belief propagation at each iteration produces a current estimate connectivity by greedily connecting each i'th node to its b_i highest-belief neighbors, we consider the adjacency matrix of an unconverged estimate $\mathbf{A}$, which sums to the target degree across rows, but not necessarily across columns. A key criterion that determines the correctness of the current estimate is the symmetry of estimated matrix $\mathbf{A}$. In other words, similarly to the pairwise consistency constraints in the b-matching Markov random field, a node's beliefs cannot be correct if it connects to nodes that do not connect back to it. Treating $\mathbf{A}$ as a partial directed adjacency, such that A_{ij} represents whether a directed edge exists from node i to node j, the symmetry of $\mathbf{A}$ is equivalently viewed as whether all directed edges are reciprocated. Let $\mathbf{A}'$ be the adjacency matrix only connecting reciprocated edges, computable as $\mathbf{A}' = \mathbf{A} \odot \mathbf{A}^\top$, where the symbol $\odot$ represents the element-wise, Hadamard

product[14]. All nodes in $\mathbf{A}'$ with unreciprocated edges have deficient degrees, needing additional edges to reach their target degree. Let S be the index set of all nodes with deficient degree

$$S = \left\{ i \middle| \sum_j A'_{ij} < b_i \right\}.$$

A feasible b-matching is constructed by iteratively choosing two nodes from S, connecting them, and checking if they still belong to set S. Once a node reaches its target degree, it is removed from S. Natural heuristics for choosing the sequence of edges to add include greedily choosing the maximum weight candidate, or randomly choosing an edge.

[14] Alternately, since $\mathbf{A}$ is binary, an element-wise logical AND operation is an equivalent computation.

Appendix B

Additional Proofs

B.1 Proof of convergence for bipartite b-matching with belief propagation

Theorem 4.2.1 *Given bipartite input graph $G = \{V, E, w, b\}$, such that a unique, optimal b-matching $\hat{E}$ has weight greater by constant ϵ than any other b-matching, i.e.,*

$$\epsilon \leq w(\hat{E}) - \max_{E' \in M, E' \neq \hat{E}} w(E'),$$

and all weights are bounded inclusively between $w_{\min}$ and $w_{\max}$, i.e.,

$$w_{\min} \leq w(e) \leq w_{\max}, \forall e \in E,$$

the maximum belief at iteration t for any node $v \in V$ indicates the corresponding neighbors in true optimum $\hat{E}$ when $t = \mathcal{O}(|V|)$.

Proof. Using the unwrapped graph construction for the b-matching graphical model, the theorem is true if and only if the root node's connectivity in the maximum weight b-matching of node v's unwrapped graph of height $h = \mathcal{O}(|V|)$ maps to the true optimal b-matching $\hat{E}$. The theorem can then be proved by contradiction. Assume that an unwrapped graph T_v for node v of any height $h \geq |V|(w_{\max} - w_{\min})/2\epsilon$ has a maximum weight b-matching such that root node r connects to nodes that do not map to the b-matched neighbors of v in optimal b-matching $\hat{E}$.

Let $\hat{E}^T$ be the set of edges in the maximum weight b-matching of T_v, and let $\hat{E}^G$ be the reverse-mapping of true optimal b-matching $\hat{E}$ to unwrapped graph T_v, i.e., the b-matching that consists

of all edges in T_v that map to the edges in $\hat{E}$. These two b-matchings of T_v observe the following lemma, which, like each lemma in this proof, is proved later.

Lemma B.1.1. *For any two distinct b-matchings $\hat{E}^T$ and $\hat{E}^G$ on tree T_v, such that the b-matched edges for the root node differ between $\hat{E}^T$ and $\hat{E}^G$, there is an alternating path P on T_v from a leaf to the root then to another leaf that alternates between edges exclusively in each b-matching. I.e., for any adjacent edges $e_i, e_j \in P$, either $e_1 \in \hat{E}^T \setminus \hat{E}^G$ and $e_2 \in \hat{E}^G \setminus \hat{E}^T$ or vice versa.*

Let P_T be an alternating path between $\hat{E}^T$ and $\hat{E}^G$. Let P_G be the path in G composed of the mapped edges of P_T, which obeys the following lemma.

Lemma B.1.2. *Let P_G be a path of length h on bipartite graph $G = \{V, E\}$. Path P_G can be decomposed into a set C of τ cycles such that $\tau \geq (2h-1)/|V|$, and a remainder path ρ of odd length at most $|V| - 1$.*

Let C be the set of cycles comprising P_G along with remainder ρ. Because G is bipartite, each cycle $c \in C$ is of even cardinality, and because C comes from alternating path P_G, every other edge in c is in $\hat{E}$. For each $c \in C$, a b-matching E^c can be constructed by taking optimal matching $\hat{E}$ and toggling all edges in cycle c. This produces an alternate b-matching that differs from the optimal b-matching, and thus differs in weight by at least ϵ. This in turn implies that the edges in c from optimal b-matching $\hat{E}$ have greater weight than edges in c from $\hat{E}^T$. Combining the bounded weight differences from each of the τ cycles in C,

$$w(C \cup \hat{E}) - w(C \setminus \hat{E}) \geq \tau\epsilon \geq (2h-1)\epsilon/|V|.$$

The remainder of P_G is a path of odd length, which means one end of the path is an edge from $\hat{E}$ and the other end of the path is an edge mapped from $\hat{E}^T$. A cycle c_ρ can be constructed by removing the end-edge from $\hat{E}^T$ and replace it with an edge back to the first node of ρ. This creates cycle c_ρ which is missing an edge from ρ and contains an additional edge that was not in ρ. The edges in c_ρ similarly alternate between edges in $\hat{E}$ and otherwise, so using the same construction above to bound the weight difference guarantees that

$$w(c_\rho \cap \hat{E}) - w(c_\rho \setminus \hat{E}) \geq \epsilon.$$

Accounting for the worst-case condition that the extra edge and the omitted edge are chosen adversarially, the difference in weights in ρ are bounded by

$$w(\rho \cap \hat{E}) - w(\rho \setminus \hat{E}) \geq \epsilon + w_{\min} - w_{\max}.$$

Combining these derived inequalities, edges in P_T obey the following bound:

$$w(P_G \cap \hat{E}) - w(P_T \cap \hat{E}^T) \geq 2h\epsilon/|V| + w_{\min} - w_{\max}.$$

Finally, the initial assumption stated that $h \geq |V|(w_{\max} - w_{\min})/2\epsilon$, which directly implies

$$w(P_T \cap \hat{E}^G) \geq w(P_T \cap \hat{E}^T).$$

This produces a contradiction since it implies that a b-matching of greater weight than $\hat{E}^T$, which is defined as the maximum weight b-matching of T_v, can be produced by toggling the edges of P_T. This contradiction occurs whenever $h \geq \mathcal{O}(|V|)$, and its impossibility proves the theorem using the equivalence of the unwrapped graph and loopy belief propagation max-marginals. □

Proof of Lemma B.1.1. The proof is by construction. It is convenient to construct P_G in two halves, each a path from the root to a leaf. The first half of P_T starts with any edge from $\hat{E}^T$ that is not in $\hat{E}^G$, which must exist as given in this lemma. Following the first edge, an alternating edge must exist from the current node to one of its children because, the parent-edge is in $\hat{E}^T$ but not $\hat{E}^G$, which means for current node v, $\hat{E}^T$ must contain edges from v to $b_v - 1$ children and $\hat{E}^G$ must contain edges from v to b_v children, leaving at least one edge that is in $\hat{E}^G$ that is not in $\hat{E}^T$. The same argument applies at all even depths in the tree and the reverse argument applies at all odd depths in the tree, guaranteeing an alternating path from the root to a leaf. The second half of P_G is constructed in the same way, reversing the roles of $\hat{E}^T$ and $\hat{E}^G$. Combining the two alternating paths from the root to leaves yields an alternating path from leaf to leaf. □

Proof of Lemma B.1.2. Start with $C = \emptyset$. Visit nodes in G in the order of P_G. Any time a node is visited that has been previously visited, remove the cycle from P_G, add it to C and continue visiting nodes in the rest of P_G. Note that removing cycles from P_G does not break the continuity of the path since cycles begin and end on the same node. Each cycle is at most length $|V|$, visiting every node in G once, and P_G is of length $2h - 1$ (the path has two nodes at each depth except for the one root at depth zero). Thus, the fewest number of cycles is $(2h - 1)/|V|$. The remainder path does

not contain a cycle, and thus cannot visit any node more than once, meaning it is of length at most $|V| - 1$. The remainder path is odd because all cycles removed from P_G are even, and P_G is of odd length. □

B.2 Proof of convergence for b-matching when the linear programming relaxation is tight

This section contains a summary of recent showing the relationship between belief propagation for b-matching and linear programming relaxation [Bayati *et al.*, To appear]. Only part of the known results are summarized here, and this section omits additional, analogous results about asynchronous belief propagation. The *linear programming relaxation* (LP relaxation) of a maximum weight b-matching problem defined on graph $G = \{V, E, w, b\}$ is the following optimization

$$\max_{\mathbf{A} \in [0,1]^{n \times n}} \sum_{v_i, v_j \in V} A_{ij} w(v_i, v_j) \text{ s.t. } \sum_j A_{ij} = b_{v_i}, \forall v_i \in V, A_{ij} = A_{ji}, \tag{B.1}$$

where in the formula above, the nodes in V are indexed by some arbitrary ordering so an equivalent adjacency matrix formulation is available. In other words, the relaxation of the *integer program* definition of the maximum weight b-matching. When all entries of the solution to the optimization are at the boundaries $\{0, 1\}$, the linear programming relaxation is considered *tight*. Otherwise, some entries have fractional solutions.

A useful tool for the proof of the convergence on b-matchings with tight LP relaxation is the construction of a *double-cover* graph of G, which is a bipartite graph constructed by duplicating each node in G and connecting each node to its neighbors' duplicated nodes in the opposite bipartition. Let V_1 and V_2 be two copies of node-set V. Let mapping $\mathrm{orig}(v)$ map any node from V_1 or V_2 back to its original node in V. Then $\tilde{E}$ be the set of edges

$$\tilde{E} = \{(u, v) | u \in V_1, v \in V_2, (\mathrm{orig}(u), \mathrm{orig}(v)) \in E\}.$$

Let the target degrees and weights of the edges be copied respectively into $\tilde{b}$ and $\tilde{w}$. The double-cover of G is $\tilde{G} = \{V_1 \cup V_2, \tilde{E}, \tilde{w}, \tilde{b}\}$.

The important property of double-cover $\tilde{G}$ is that running belief propagation on $\tilde{G}$ produces exactly the same updates as running belief propagation on the original graph G. Since the weights

of each edge and the local neighborhood topology at each node is identical in both G and $\tilde{G}$, the belief update rules operate on exactly the same quantities. Thus, the convergence (and any other) behavior of belief propagation on $\tilde{G}$ is identical to belief propagation on G. Since the double-cover is a bipartite graph, and its weights are bounded, it nearly fits the conditions for Theorem 4.2.1. The final criterion to guarantee belief propagation's convergence on $\tilde{G}$ (and therefore G) is that the solution is unique.

Theorem 4.2.2 *Assume that the linear programming relaxation of the b-matching problem on G has a unique, integer solution, and that the weights in G are bounded. Then belief propagation converges to the optimal solution in $\mathcal{O}(|V|)$ iterations.*

Proof. Using the double-cover construction described above, the belief propagation on the double-cover graph $\tilde{G}$ is identical to belief propagation on G. Since $\tilde{G}$ is a bipartite graph with bounded weights, if b-matching on $\tilde{G}$ has a unique solution that corresponds to the unique solution of G, then $\tilde{G}$ fits the conditions of Theorem 4.2.1, and is thus guaranteed to converge to the correct solution.

Consider any b-matching on $\tilde{G}$ with total weight $\tilde{w}$. The following Lemma relates b-matchings on $\tilde{G}$ with solutions to the LP in Equation (B.1).

Lemma B.2.1. *Given any b-matching on $\tilde{G}$ with weight $\tilde{w}$, a feasible solution $\tilde{\mathbf{A}}$ to the linear program in Equation (B.1) can be constructed that has weight $\tilde{w}/2$.*

If the unique, integral solution of the LP relaxation is $w*$, the b-matching constructed by copying the integral LP relaxation solution onto $\tilde{G}$ has weight $2w*$. According to Lemma B.2.1, all feasible b-matchings in $\tilde{G}$ have equivalent feasible LP solutions, which means, since the LP solution with weight $w*$ is unique, any other feasible b-matching must have weight less than $w*$. □

Proof of Lemma B.2.1. Let $\hat{E}$ be the b-matching on $\tilde{G}$. Consider the adjacency matrix of $\hat{E}$, which, if nodes in V_1 are indexed in the first $|V|$ rows and nodes in V_2 are indexed in the last $|V|$ rows, is block-off-diagonal. Let $\mathbf{A}$ be the upper right quadrant of the b-matching adjacency matrix, which is then structured as the block matrix

$$\begin{bmatrix} \mathbf{0} & \mathbf{A} \\ \mathbf{A}^\top & \mathbf{0} \end{bmatrix}.$$

By the b-matching constraints, the rows and columns of $\mathbf{A}$ sum to b, but $\mathbf{A}$ is not necessarily symmetric. Construct a feasible solution to the LP relaxation by averaging $\mathbf{A}$ and $\mathbf{A}^\top$, $\tilde{\mathbf{A}} = \frac{1}{2}(\mathbf{A} + \mathbf{A}^\top)$. Simple algebra now verifies that $\tilde{A}$ has objective value $\tilde{w}/2$ and is a feasible solution to the objective in Equation (B.1). □

In general, it is not trivial to identify whether a given b-matching problem has a tight linear programming relaxation. For maximum weight 1-matchings, a necessary and sufficient condition for the tightness of the LP relaxation is that the *line graph* of the input is a *perfect graph* [Jebara, 2009]. However, even though a polynomial time algorithm exists to test for graph perfection [Chudnovsky *et al.*, 2005], current implementations of the algorithm have a high order polynomial running time and testing whether a graph is perfect is expensive in practice.

B.3 Proof of collaborative filtering data concentration and corollaries

Theorem 6.2.1 *Given that users $U = \{u_1, \ldots, u_m\}$ and rated items $V = \{v_1, \ldots, v_n\}$ are drawn* iid *from arbitrary distributions $\mathbb{D}_u$ and $\mathbb{D}_v$ and that the probability of positive rating by a user for an item is determined by a function $g(u_i, v_j) \mapsto [0, 1]$, the average of query ratings by each user is concentrated around the average of his or her training ratings. Formally,*

$$\begin{aligned} \Pr\left(\Delta(X_i) \geq \epsilon\right) &\leq 2\exp\left(-\frac{\epsilon^2 m_i \hat{m}_i}{2(m_i + \hat{m}_i)}\right), \\ \Pr\left(\Delta(X_i) \leq -\epsilon\right) &\leq 2\exp\left(-\frac{\epsilon^2 m_i \hat{m}_i}{2(m_i + \hat{m}_i)}\right). \end{aligned} \tag{B.2}$$

Proof of Theorem 6.2.1. McDiarmid's Inequality bounds the deviation probability of a function over independent (but not necessarily identical) random variables from its expectation in terms of its Lipschitz constants [McDiarmid, 1989], which are the maximum change in the function value induced by changing any input variable. The Lipschitz constants for function Δ are $\ell_j = 1/m_i$ for $1 \leq j \leq m_i$, and $\ell_j = 1/\hat{m}_i$ otherwise. Although the rating random variables are not identically distributed, they are independently sampled, so we can apply McDiarmid's Inequality (and simplify) to obtain

$$\Pr\left(\Delta(\bar{X}_i) - \mathbb{E}[\Delta] \geq t_1\right) \leq \exp\left(-\frac{2t^2 m_i \hat{m}_i}{m_i + \hat{m}_i}\right) \tag{B.3}$$

The left-hand side quantity inside the probability contains $\mathbb{E}[\Delta]$, which should be close to zero, but not exactly zero (if it were zero, Eq. B.3 would be the bound). Since the probability of X_{ij} is a function of u_i and v_j, the expectation is

$$\begin{aligned}\mathbb{E}\left[\Delta(\bar{Y}_i)\right] &= \mathbb{E}\left[\frac{1}{m_i}\sum_{j|ij\in T} X_{ij} - \frac{1}{\hat{m}_i}\sum_{j|ij\in Q} X_{ij}\right] \\ &= \frac{1}{m_i}\sum_{j|ij\in T} g(u_i, v_j) - \frac{1}{\hat{m}_i}\sum_{j|\in Q} g(u_i, v_j) \\ &\stackrel{\text{def}}{=} D(V)\end{aligned}$$

We define the expected deviation above as a function over the items $V = \{v_j | ij \in T \cup Q\}$, denoted $D(V)$ for brevity. Because this analysis is of one user's ratings, we can treat the user input u_i to $g(u_i, v_j)$ as a constant. Since the range of the probability function $g(u_i, v_i)$ is $[0, 1]$, the Lipschitz constants for $D(V)$ are $\ell_j = 1/m_i$ for $1 \leq j \leq m_i$, and $\ell_j = 1/\hat{m}_i$ otherwise. We apply McDiarmid's Inequality again.

$$\Pr\left(D(V) - \mathbb{E}[D(V)] \geq t_2\right) \leq \exp\left(-\frac{2t_2^2 m_i \hat{m}_i}{m_i + \hat{m}_i}\right).$$

The expectation of $D(V)$ can be written as the integral

$$\mathbb{E}[D(V)] = \int \Pr(\{v_j | ij \in T \cup Q\}) D(V) dV.$$

Since the v's are *iid*, the integral decomposes into

$$\begin{aligned}\mathbb{E}[D(V)] &= \frac{1}{m_i}\sum_{j|ij\in T}\int \Pr(v_j) g(u_i, v_j) dv_j \\ &\quad -\frac{1}{\hat{m}_i}\sum_{j|ij\in Q}\int \Pr(v_j) g(u_i, v_j) dv_j.\end{aligned}$$

Since each $\Pr(v_j) = \Pr(v)$ for all j, by a change of variables all integrals above are identical. The expected value $\mathbb{E}[D(V)]$ is therefore zero. This leaves a bound on the value of $D(V)$.

$$\Pr\left(D(V) \geq t_2\right) \exp\left(-\frac{2t_2^2 m_i \hat{m}_i}{m_i + \hat{m}_i}\right)$$

To combine the bounds, we define a quantity to represent the probability of each deviation. First, let the probability of $D(V)$ exceeding some constant t_2 be $\frac{\delta}{2}$.

$$\frac{\delta}{2} = \exp\left(-\frac{2t_2^2 m_i \hat{m}_i}{m_i + \hat{m}_i}\right)$$

Second, let the probability of $\Delta(\bar{X}_i)$ exceeding its expectation by more than a constant t also be $\frac{\delta}{2}$,

$$\frac{\delta}{2} = \exp\left(-\frac{2t^2 m_i \hat{m}_i}{m_i + \hat{m}_i}\right).$$

We can write both t and t_2 in terms of δ:

$$t_1 = t_2 = \sqrt{\frac{m_i + \hat{m}_i}{2m_i\hat{m}_i \log\frac{2}{\delta}}}.$$

Define ϵ as the concatenation of deviations t_1 and t_2,

$$\epsilon = t_1 + t_2 = 2\sqrt{\frac{m_i + \hat{m}_i}{2m_i\hat{m}_i \log\frac{2}{\delta}}}.$$

By construction, the total deviation ϵ occurs with probability greater than δ. Solving for δ provides the final bound in Equation B.2. The bound in the other direction follows easily since McDiarmid's Inequality is also symmetric. □

Although the above analysis refers only to the ratings of the user, the generative model we describe is symmetric between users and items. Similar analysis therefore applies directly to item ratings as well.

Corollary B.3.1. *Under the same assumptions as Theorem 6.2.1, the average of query ratings for each item is concentrated around the average of its training ratings.*

Additionally, even though Theorem 6.2.1 specifically concerns preference graphs, it can be easily extended to show the concentration of edge connectivity in general unipartite and bipartite graphs as follows.

Corollary B.3.2. *The concentration bound in Theorem 6.2.1 applies to general graphs; assuming that edges and non-edges are revealed randomly, nodes are generated* iid *from some distribution and the probability of an edge is determined by a function of its vertices, the average connectivity of unobserved (testing) node-pairs is concentrated around the average connectivity of observable (training) node-pairs. The probability of deviation is bounded by the same formula as in Theorem 6.2.1.*

B.4 Concentration proof for rating features

The concentration proof for rating features uses some similar concepts to the proof of the binary ratings in the Appendix B.3, but the target quantity being bounded in this section is the deviation between the empirical average of the training ratings and the expected average of the query ratings.

Theorem 6.3.1 *For the ratings of user i, the difference*

$$\epsilon_{ik} = \frac{1}{m_i} \sum_{j|(i,j)\in T} f_k(X_{ij}) - \frac{1}{\hat{m}_i} \sum_{j|(i,j)\in Q} \mathbb{E}_{p(X_{ij}|u_i,v_j)}[f_k(X_{ij})]$$

between the average of $f_k(X) \mapsto [0,1]$ over the observed ratings and the average of the expected value of $f_k(X)$ over the query ratings is bounded above by

$$\epsilon_{ik} \leq \sqrt{\frac{\ln \frac{2}{\delta}}{2m_i}} + \sqrt{\frac{(m_i + \hat{m}_i) \ln \frac{2}{\delta}}{2m_i \hat{m}_i}}$$

with probability $1 - \delta$.

Proof of Theorem 6.3.1. The main intuition is that the total deviation between the expected query average and training average is composed of (1) the deviation between the training average and the expected training average and (2) the deviation between the expected training average and the expected query average. Since each component involves independent (though not necessarily *iid*) variables, McDiarmid's inequality [McDiarmid, 1989] can be invoked.

Recall the deviation of interest,

$$\epsilon_{ik} = \frac{1}{m_i} \sum_{j|(i,j)\in T} f_k(x_{ij}) - \frac{1}{\hat{m}_i} \sum_{j|(i,j)\in Q} \sum_{x_{ij}} f_k(x_{ij}) p(x_{ij}|u_i, v_j).$$

Clearly, ϵ_{ik} is a function of the independent ratings x_{ij} for all j such that $(i,j) \in T$ and a function of the independent item descriptors v_j for all j such that $(i,j) \in Q$. The Lipschitz constants of this function of two sets of independent variables will be examined.

Since the range of function f_k is bounded by $[0,1]$, the deviation function ϵ_{ik} is Lipschitz continuous with constants $1/m_i$ for the training ratings and $1/\hat{m}_i$ for the query item variables. Furthermore, ϵ_{ik} is a function of two sets of independent variables allowing the application of McDiarmid's inequality (twice). After simplifying, the probability of ϵ_{ik} exceeding its expected value by a constant t_1 is bounded by

$$p\left(\epsilon_{ik} - \mathbb{E}_{x,v}[\epsilon_{ik}] \geq t_1\right) \leq \exp\left(-\frac{2m_i \hat{m}_i t_1^2}{m_i + \hat{m}_i}\right). \tag{B.4}$$

Here, we write $\mathbb{E}_{x,v}$ to denote the expectation over the training ratings $\{x_{ij}|\ (i,j) \in\ T\}$ and all item descriptors, $\{v_j|(i,j) \in T \cup Q\}$. The expectation $\mathbb{E}[\epsilon_{ik}]$ is not exactly zero but can be shown to be close to zero with high probability. First, simplify the quantity using the linearity of expectation to obtain

$$\mathbb{E}_{x,v}[\epsilon_{ik}] \quad = \quad \frac{1}{m_i} \sum_{j|(i,j)\in T} \mathbb{E}_x[f_k(x_{ij})] - \frac{1}{\hat{m}_i} \sum_{j|(i,j)\in Q} \mathbb{E}_v \left[\sum_x f_k(x_{ij}) p(x|u_i, v_j) \right].$$

Rewrite the training expectation directly in terms of the training probabilities $\sum_{x_{ij}} f_k(x_{ij}) p(x_{ij}|u_i, v_j)$. Similarly, since all the v variables are sampled *iid*, rewrite their expectation explicitly as follows

$$\begin{aligned} \mathbb{E}_{x,v}[\epsilon_{ik}] \quad = \quad & \frac{1}{m_i} \sum_{j|(i,j)\in T} \sum_{x_{ij}} f_k(x_{ij}) p(x_{ij}|u_i, v_j) - \\ & \frac{1}{\hat{m}_i} \sum_{j|(i,j)\in Q} \int_v \sum_{x_{ij}} f_k(x_{ij}) p(x_{ij}|u_i, v) p(v) dv. \end{aligned}$$

Since the query summation no longer depends on the j index, omit the average over the j query indices,

$$\mathbb{E}_{x,v}[\epsilon_{ik}] \quad = \quad \frac{1}{m_i} \sum_{j|(i,j)\in T} \sum_{x_{ij}} f_k(x_{ij}) p(x_{ij}|u_i, v_j) - \int_v \sum_x f_k(x) p(x|u_i, v) p(v) dv. \quad \text{(B.5)}$$

After the simplifications illustrated above, the training sum (the first term in Equation (B.5)) is a function of the training item descriptors v_j for all j where $(i,j) \in T$. This function has Lipschitz constant $1/m_i$. Also, the second term in Equation (B.5) is the expectation of the function. Therefore, McDiarmid's inequality directly applies to this difference. The probability of $\mathbb{E}[\epsilon_{ik}]$ exceeding a constant t_2 is bounded by

$$p\left(\mathbb{E}[\epsilon_{ik}] \geq t_2\right) \leq \exp\left(-2 m_i t_2^2\right). \quad \text{(B.6)}$$

A union bound will be used to combine both deviations. Define the right-hand side of Equation (B.4) as

$$\frac{\delta}{2} = \exp\left(-\frac{2 m_i \hat{m}_i t_1^2}{m_i + \hat{m}_i} \right).$$

Rewriting the above such that the corresponding deviation t_1 is a function of δ yields

$$t_1 = \sqrt{\frac{(m_i + \hat{m}_i) \ln \frac{2}{\delta}}{2 m_i \hat{m}_i}}.$$

Similarly, let the right-hand side of the deviation bound in Equation (B.6) be $\delta/2$. The corresponding deviation as a function of δ is then

$$t_2 = \sqrt{\frac{\ln \frac{2}{\delta}}{2m_i}}.$$

Defining the total deviation as $\epsilon_{ik} = t_1 + t_2$ and applying a union bound completes the proof. $\square$

Appendix C

Additional Algorithm Derivations

C.1 Maximum entropy dual for collaborative filtering model

Since the number of queries can be much larger than the number of users and items, convert the maximum entropy problem into dual form via the Lagrangian

$$\begin{aligned}
&\min_{\gamma,\lambda\in\mathbb{R}^+,\zeta\in\mathbb{R}} \max_{p} \sum_{ij\in Q} H(p_{ij}(X_{ij}))+ \\
&\quad \sum_{\substack{ij\in Q \\ X_{ij}}} p_{ij}(X_{ij}) \ln p_0(X_{ij}) - \sum_{ij\in Q} \zeta_{ij}(\sum_{X_i} p_{ij}(X_{ij}) - 1) \\
&\quad + \sum_{k,i} (\gamma^+_{ik} - \gamma^-_{ik}) \left(\frac{1}{\hat{m}_i} \sum_{j|(i,j)\in Q} \mathbb{E}_{X_{ij}}[f_k(X_{ij})] - \mu_{ik} \right) \\
&\quad + \sum_{k,j} (\lambda^+_{jk} - \lambda^-_{jk}) \left(\frac{1}{\hat{n}_j} \sum_{i|(i,j)\in Q} \mathbb{E}_{X_{ij}}[f_k(X_{ij})] - \nu_{jk} \right) \\
&\quad + (\gamma^+_{ik} + \gamma^-_{ik})\alpha_i + (\lambda^+_{jk} + \lambda^-_{jk})\beta_j.
\end{aligned}$$

Above, we define $\mathbb{E}_{X_{ij}}[f_k(X_{ij})] = \sum_{X_{ij}} p_{ij}(X_{ij}) f_k(X_{ij})$ and use $p_{ij}(X_{ij})$ as shorthand for the conditional probability $p(X_{ij}|u_i, v_j)$. The Lagrange multipliers $\gamma^{\pm}_{ik}$ and $\lambda^{\pm}_{jk}$ correspond to the positive and negative absolute value constraints for the user and item averages. The ζ_{ij} multipliers correspond to equality constraints that force distributions to normalize.

The probabilities and the normalization multipliers can be solved for analytically resulting in

the much simpler dual minimization program $\min_{\gamma,\lambda\geq 0}\mathcal{D}$ where the dual cost function is given by

$$\begin{aligned}\mathcal{D} &= \sum_{ik}(\gamma_{ik}^{+}+\gamma_{ik}^{-})\alpha_i-(\gamma_{ik}^{+}-\gamma_{ik}^{-})\mu_{ik}+\\ &\quad \sum_{kj}(\lambda_{jk}^{+}+\lambda_{jk}^{-})\beta_j-(\lambda_{jk}^{+}-\lambda_{jk}^{-})\nu_{jk}+\sum_{ij\in Q}\ln Z_{ij}.\end{aligned}$$

Here, Z_{ij} is the normalizing partition function for the estimated distribution for rating X_{ij}, and is defined as

$$Z_{ij}=\sum_{X_{ij}}p_0(X_{ij})\exp\left(\sum_k\left(\frac{\gamma_{ik}^{+}-\gamma_{ik}^{-}}{\hat{m}_i}+\frac{\lambda_{jk}^{+}-\lambda_{jk}^{-}}{\hat{n}_j}\right)f_k(x_{ij})\right).$$

Once the Lagrange multipliers are found, the estimated probabilities are normalized Gibbs distributions of the form

$$p_{ij}(X_{ij})\propto p_0(X_{ij})\exp\left(\sum_k\left(\frac{\gamma_{ik}^{+}-\gamma_{ik}^{-}}{\hat{m}_i}+\frac{\lambda_{jk}^{+}-\lambda_{jk}^{-}}{\hat{n}_j}\right)f_k(X_{ij})\right).$$

Optimizing the cost function $\mathcal{D}$ requires taking partial derivatives which can be written in terms of normalized probabilities as follows

$$\begin{aligned}\frac{\partial\mathcal{D}}{\partial\gamma_{ik}^{\pm}} &= \alpha_i\mp\mu_{ik}\pm\frac{1}{\hat{m}_i}\sum_{j|(i,j)\in Q}\sum_{X_{ij}}p_{ij}(X_{ij})f_k(X_{ij})\\ \frac{\partial\mathcal{D}}{\partial\lambda_{jk}^{\pm}} &= \beta_j\mp\nu_{jk}\pm\frac{1}{\hat{n}_j}\sum_{i|(i,j)\in Q}\sum_{X_{ij}}p_{ij}(X_{ij})f_k(X_{ij}),\end{aligned}$$

where the $\pm$ and $\mp$ symbols are shorthand to indicate the respective signs for each component of the gradient for the multipliers of positive and negative constraints.

Bibliography

[Airoldi *et al.*, 2008] E. Airoldi, D. Blei, S. Fienberg, and E. Xing. Mixed membership stochastic blockmodels. *J. Mach. Learn. Res.*, 9:1981–2014, 2008.

[Albert and Barabási, 2002] R. Albert and A. Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–97, Jan 2002.

[Asuncion and Newman, 2007] A. Asuncion and D. Newman. UCI machine learning repository. University of California, Irvine, School of Information and Computer Sciences. http://www.ics.uci.edu/∼mlearn/MLRepository.html, 2007.

[Barabási, 2003] A. Barabási. Linked: The new science of networks. *J. Artificial Societies and Social Simulation*, 6(2), 2003.

[Bayati *et al.*, 2005] M. Bayati, D. Shah, and M. Sharma. Maximum weight matching via max-product belief propagation. In *Proc. of the IEEE International Symposium on Information Theory*, 2005.

[Bayati *et al.*, 2007] M. Bayati, C. Borgs, J. Chayes, and R. Zecchina. Belief-propagation for weighted b-matchings on arbitrary graphs and its relation to linear programs with integer solutions. *CoRR*, abs/0709.1190, 2007.

[Bayati *et al.*, 2008] M. Bayati, D. Shah, and M. Sharma. Max-product for maximum weight matching: Convergence, correctness, and LP duality. *IEEE Transactions on Information Theory*, 54(3):1241–1251, 2008.

[Bayati *et al.*, To appear] M. Bayati, C. Borgs, J. Chayes, and R. Zecchina. Belief-propagation for weighted b-matchings on arbitrary graphs and its relation to linear programs with integer solutions. *SIAM Journal on Discrete Mathematics*, To appear.

[Bell and Koren, 2007] R. Bell and Y. Koren. Lessons from the netflix prize challenge. *SIGKDD Explorations Newsletter*, 9(2):75–79, December 2007.

[Belongie *et al.*, 2002] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:509–522, 2002.

[Bentley, 1975] J. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18:509–517, September 1975.

[Blei *et al.*, 2003] D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

[Boyd and Ellison, 2007] Danah M. Boyd and Nicole B. Ellison. Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*, 13(1):article 11, October 2007.

[Breese *et al.*, 1998] J. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings UAI 1998*. Morgan Kaufmann, 1998.

[Caetano *et al.*, 2009] T. Caetano, J. McAuley, L. Cheng, Q. Le, and A. Smola. Learning graph matching. *IEEE Trans. Patt. Analysis and Mach. Intel. (PAMI)*, 31(6):1048–1058, Jun 2009.

[Chang and Blei, 2010] J. Chang and D. Blei. Hierarchical relational models for document networks. *Annals of Applied Statistics*, 4:124–150, 2010.

[Chang *et al.*, 2009] J. Chang, J. Boyd-Graber, and D. Blei. Connections between the lines: augmenting social networks with text. In J. Elder IV, F. Fogelman-Soulié, P. Flach, and M. Zaki, editors, *KDD*, pages 169–178. ACM, 2009.

[Chechik *et al.*, 2010] G. Chechik, V. Sharma, U. Shalit, and S. Bengio. Large scale online learning of image similarity through ranking. *J. Mach. Learn. Res.*, 11:1109–1135, March 2010.

[Christakis and Fowler, 2011] N. Christakis and J. Fowler. *Connected: The Surprising Power of Our Social Networks and How They Shape Our Lives*. Little, Brown and Company, 2011.

[Chudnovsky *et al.*, 2005] M. Chudnovsky, G. Cornuéjols, X. Liu, P. Seymour, and K. Vuskovic. Recognizing berge graphs. *Combinatorica*, 25(2):143–186, 2005.

[Conway *et al.*, 1987] J. Conway, N. Sloane, and E. Bannai. *Sphere-packings, lattices, and groups*. Springer-Verlag New York, Inc., New York, NY, USA, 1987.

[Cormen *et al.*, 2001] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to algorithms*. McGraw-Hill Book Company, Cambridge, London, 2 edition, 2001.

[Cuturi, 2007] M. Cuturi. Permanents, transportation polytopes and positive definite kernels on histograms. In *International Joint Conference on Artificial Intelligence, IJCAI*, 2007.

[Dagum and Luby, 1992] P. Dagum and M. Luby. Approximating the permanent of graphs with large factors. *Theoretical Computer Science*, 102(2):283–305, 1992.

[De Francisci Morales *et al.*, 2011] G. De Francisci Morales, A. Gionis, and M. Sozio. Social content matching in mapreduce. *Proc. VLDB Endow.*, 4:460–469, April 2011.

[Devroye and Wagner, 1979] L. Devroye and T. Wagner. Distribution-free performance bounds for potential function rules. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1979.

[Duan and Pettie, 2010] R. Duan and S. Pettie. Approximating maximum weight matching in near-linear time. In *Proceedings 51st IEEE Symposium on Foundations of Computer Science (FOCS)*, 2010.

[Dudík *et al.*, 2007] M. Dudík, D. Blei, and R. Schapire. Hierarchical maximum entropy density estimation. In Z. Ghahramani, editor, *Proceedings of the ICML 2007*, pages 249–256. Omnipress, 2007.

[Edmonds, 1965] J Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[Erdos and Renyi, 1959] P. Erdos and A. Renyi. On random graphs i. *Publicationes Mathematicae*, 1959.

[Fagin *et al.*, 2003] R. Fagin, R. Kumar, and D. Sivakumar. Comparing top k lists. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 2003.

[Fremuth-Paeger and Jungnickel, 1999] C. Fremuth-Paeger and D. Jungnickel. Balanced network flows. i. a unifying framework for design and analysis of matching algorithms. *Networks*, 33(1), 1999.

[Frey and Dueck, 2006] B. Frey and D. Dueck. Mixture modeling by affinity propagation. In *Advances in Neural Information Processing Systems 18*, pages 379–386. MIT Press, 2006.

[Gabow and Tarjan, 1989] H. Gabow and R. Tarjan. Faster scaling algorithms for network problems. *SIAM J. Comput.*, 18(5):1013–1036, 1989.

[Gamarnik *et al.*, 2010] D. Gamarnik, D. Shah, and Y. Wei. Belief propagation for min-cost network flow: convergence & correctness. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 279–292, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.

[Gelman *et al.*, 2003] A. Gelman, J. Carlin, H. Stern, and D. Rubin. *Bayesian Data Analysis, Second Edition*. Chapman & Hall/CRC, July 2003.

[Getoor and Taskar, 2007] L. Getoor and B. Taskar. *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2007.

[Gionis *et al.*, 1999] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[Givoni and Frey, 2009] I. Givoni and B. Frey. A binary variable model for affinity propagation. *Neural Computation*, 21(6):1589–1600, 2009.

[Goldberg and Kennedy, 1995] A. Goldberg and R. Kennedy. An efficient cost scaling algorithm for the assignment problem. *Math. Program.*, pages 153–177, 1995.

[Goldberg *et al.*, 2001] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Inf. Retr.*, 4(2), 2001.

[Heskes, 2003] T. Heskes. Stable fixed points of loopy belief propagation are local minima of the Bethe free energy. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 343–350. MIT Press, Cambridge, MA, 2003.

[Heskes, 2004] Tom Heskes. On the uniqueness of loopy belief propagation fixed points. *Neural Comput.*, 16(11):2379–2413, 2004.

[Heskes, 2006] T. Heskes. Convexity arguments for efficient minimization of the Bethe and Kikuchi free energies. *Journal of Artificial Intelligence Research*, 26, 2006.

[Ho *et al.*, 2011] Q. Ho, A. Parikh, L. Song, and E. Xing. Multiscale community blockmodel for network exploration. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011.

[Hofmann and Puzicha, 1999] T. Hofmann and J. Puzicha. Latent class models for collaborative filtering. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, IJCAI '99, pages 688–693, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[Huang and Jebara, 2007] B. Huang and T. Jebara. Loopy belief propagation for bipartite maximum weight b-matching. In M. Meila and X. Shen, editors, *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics*, volume 2 of JMLR: W&CP, March 2007.

[Huang and Jebara, 2009a] B. Huang and T. Jebara. Exact graph structure estimation with degree priors. In M. Wani, M. Kantardzic, V. Palade, L. Kurgan, and Y. Qi, editors, *International Conference on Machine Learning and Applications*. IEEE Computer Society, 2009.

[Huang and Jebara, 2009b] Bert Huang and Tony Jebara. Approximating the permanent with belief propagation. *CoRR*, abs/0908.1769, 2009.

[Huang and Jebara, 2010] B. Huang and T. Jebara. Collaborative filtering via rating concentration. In Y. Teh and M. Titterington, editors, *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, volume 9 of JMLR: W&CP, May 2010.

[Huang and Jebara, 2011] B. Huang and T. Jebara. Fast b-matching via sufficient selection belief propagation. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011.

[Jebara and Shchogolev, 2006] T. Jebara and V. Shchogolev. B-matching for spectral clustering. In J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, editors, *ECML*, volume 4212 of *Lecture Notes in Computer Science*, pages 679–686. Springer, 2006.

[Jebara *et al.*, 2009] T. Jebara, J. Wang, and S. Chang. Graph construction and b-matching for semi-supervised learning. In *International Conference on Machine Learning*, 2009.

[Jebara, 2009] T. Jebara. Map estimation, message passing, and perfect graphs. In *Uncertainty in Artificial Intelligence*, 2009.

[Jerrum *et al.*, 2004] M. Jerrum, A. Sinclair, and E. Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *J. ACM*, 51(4):671–697, 2004.

[Joachims *et al.*, 2009] T. Joachims, T. Finley, and C. Yu. Cutting-plane training of structural svms. *Machine Learning*, 77(1):27–59, 2009.

[Joachims, 2006] T. Joachims. Training linear SVMs in linear time. In *ACM SIG International Conference On Knowledge Discovery and Data Mining (KDD)*, pages 217 – 226, 2006.

[Jonker and Volgenant, 1979] R. Jonker and A. Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4):225–340, 1979.

[Kolmogorov, 2009] V. Kolmogorov. Blossom V: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1:43–67, 2009. 10.1007/s12532-009-0002-8.

[Koo *et al.*, 2007] T. Koo, A. Globerson, X. Carreras, and M. Collins. Structured prediction models via the matrix-tree theorem. In *In EMNLP-CoNLL*, 2007.

[Koren *et al.*, 2009] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[LeCun *et al.*, 2001] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Intelligent Signal Processing*, pages 306–351. IEEE Press, 2001.

[Leskovec *et al.*, 2010] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani. Kronecker graphs: An approach to modeling networks. *Journal of Machine Learning Research*, 11:985–1042, 2010.

[Liben-Nowell and Kleinberg, 2007] D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *J. Am. Soc. Inf. Sci. Technol.*, 58:1019–1031, May 2007.

[Lim and Teh, 2007] Y. Lim and Y. Teh. Variational bayesian approach to movie rating prediction. In *Proceedings of KDD Cup and Workshop*, 2007.

[Maneewongvatana and Mount, 1999] S. Maneewongvatana and D. Mount. Analysis of approximate nearest neighbor searching with clustered point sets. *CoRR*, cs.CG/9901013, 1999.

[Marlin *et al.*, 2007] B. Marlin, R. Zemel, S. Roweis, and M. Slaney. Collaborative filtering and the missing at random assumption. In *Proceedings of UAI 2007*, 2007.

[Marlin, 2004] B. Marlin. Modeling user rating profiles for collaborative filtering. In *Advances in Neural Information Processing Systems 17*. MIT Press, 2004.

[Massa and Avesani, 2005] P. Massa and P. Avesani. Controversial users demand local trust metrics: An experimental study on epinions.com community. In M. Veloso and S. Kambhampati, editors, *AAAI*. MIT Press, 2005.

[Matusov *et al.*, 2004] E. Matusov, R. Zens, and H. Ney. Symmetric word alignments for statistical machine translation. In *COLING '04: Proceedings of the 20th international conference on Computational Linguistics*, page 219, Morristown, NJ, USA, 2004. Association for Computational Linguistics.

[McAuley and Caetano, 2009] J. McAuley and T. Caetano. Faster algorithms for max-product message-passing. *CoRR*, abs/0910.3301, 2009.

[McAuley and Caetano, 2010] J. McAuley and T. Caetano. Exploiting data-independence for fast belief-propagation. In J. Fürnkranz and T. Joachims, editors, *ICML*, pages 767–774. Omnipress, 2010.

[McDiarmid, 1989] C. McDiarmid. On the method of bounded differences. *Surveys in Combinatorics*, pages 148–188, 1989.

[Mehta *et al.*, 2007] A. Mehta, A. Saberi, U. Vazirani, and V. Vazirani. Adwords and generalized online matching. *J. ACM*, 54(5), 2007.

[Miller *et al.*, 2009] K. Miller, T. Griffiths, and M. Jordan. Nonparametric latent feature models for link prediction. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1276–1284. MIT Press, 2009.

[Montanari *et al.*, 2009] A. Montanari, R. Keshavan, and S. Oh. Matrix completion from a few entries. In *ISIT 2009. IEEE International Symposium on Information Theory*, 2009.

[Morris and Frey, 2003] Q. Morris and B. Frey. Denoising and untangling graphs using degree priors. In *Advances in Neural Information Processing Systems 16*. MIT Press, 2003.

[Munkres, 1957] J. Munkres. Algorithms for the Assignment and Transportation Problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[Newman *et al.*, 2001] M. Newman, S. Strogatz, and D. Watts. Random graphs with arbitrary degree distributions and their applications. *Physical Review E*, 64, 2001.

[Pearl, 1988] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[Ratliff *et al.*, 2007] N. Ratliff, J. Bagnell, and M. Zinkevich. (online) subgradient methods for structured prediction. In *Artificial Intelligence and Statistics*, volume 2 of JMLR: W&CP, 2007.

[Rennie and Srebro, 2005] J. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In L. De Raedt and S. Wrobel, editors, *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, volume 119 of *ACM International Conference Proceeding Series*, pages 713–719. ACM, 2005.

[Rennie, 2007] J. Rennie. *Extracting information from informal communication*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2007.

[Robins *et al.*, 1999] G. Robins, P. Pattison, and S. Wasserman. Logit models and logistic regressions for social networks, iii: Valued relations. *Psychometrika*, 64:371–394, 1999.

[Robins *et al.*, 2006] G. Robins, T. Snijders, P. Wang, and M. Handcock. Recent developments in exponential random graph (p*) models for social networks. *Social Networks*, 29:192–215, 2006.

[Robins *et al.*, 2007] G. Robins, P. Pattison, Y. Kalish, and D. Lusher. An introduction to exponential random graph (p*) models for social networks. *Social Networks*, 29(2):173–191, 2007.

[Ryser, 1963] H. Ryser. Combinatorial mathematics. *The Carus Mathematical Monographs*, 1963.

[Salakhutdinov and Mnih, 2008a] R. Salakhutdinov and A. Mnih. Bayesian probabilistic matrix factorization using Markov chain Monte Carlo. In *Proceedings of the International Conference on Machine Learning*, volume 25, 2008.

[Salakhutdinov and Mnih, 2008b] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems*, volume 20, 2008.

[Salez and Shah, 2009] J. Salez and D. Shah. Optimality of belief propagation for random assignment problem. In C. Mathieu, editor, *SODA*, pages 187–196. SIAM, 2009.

[Sanghavi *et al.*, 2007] S. Sanghavi, D. Malioutov, and A. Willsky. Linear programming analysis of loopy belief propagation for weighted matching. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1273–1280, Cambridge, MA, 2007. MIT Press.

[Sankowski, 2009] P. Sankowski. Maximum weight bipartite matching in matrix multiplication time. *Theor. Comput. Sci.*, 410(44):4480–4488, 2009.

[Servedio and Wan, 2005] R. Servedio and A. Wan. Computing sparse permanents faster. *Inf. Process. Lett.*, 96(3):89–92, 2005.

[Shalev-Shwartz *et al.*, 2007] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for SVM. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 807–814, New York, NY, USA, 2007. ACM.

[Shalev-Shwartz *et al.*, To appear] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: Primal estimated sub-gradient solver for SVM. *Mathematical Programming*, To appear.

[Shaw and Jebara, 2007] B. Shaw and T. Jebara. Minimum volume embedding. In M. Meila and X. Shen, editors, *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics*, volume 2 of JMLR: W&CP, March 2007.

[Shaw and Jebara, 2009] B. Shaw and T. Jebara. Structure preserving embedding. In A. Danyluk, L. Bottou, and M. Littman, editors, *International Conference on Machine Learning*, volume 382 of *ACM International Conference Proceeding Series*. ACM, 2009.

[Shaw *et al.*, To appear] B. Shaw, B. Huang, and T. Jebara. Learning a distance metric from a network. In *Advances in Neural Information Processing Systems 25*, To appear.

[Shivaswamy and Jebara, 2006] P. Shivaswamy and T. Jebara. Permutation invariant SVMs. In *International Conference on Machine Learning, ICML*, 2006.

[Srebro *et al.*, 2004] N. Srebro, J. Rennie, and T. Jaakkola. Maximum-margin matrix factorization. In *Advances in Neural Information Processing Systems 17*. MIT Press, 2004.

[Tarlow *et al.*, 2010] D. Tarlow, I. Givoni, and R. Zemel. Hop-map: Efficient message passing with high order potentials. *Journal of Machine Learning Research - Proceedings Track*, 9:812–819, 2010.

[Tatikonda and Jordan, 2002] S. Tatikonda and M. Jordan. Loopy belief propagation and gibbs measures. In *In Uncertainty in Artificial Intelligence*, pages 493–500. Morgan Kaufmann, 2002.

[Traud *et al.*, 2011] A. Traud, P. Mucha, and M. Porter. Social structure of facebook networks. *CoRR*, abs/1102.2166, 2011.

[Ungar and Foster, 1998] L Ungar and D Foster. Clustering methods for collaborative filtering. In *Proceedings of the Workshop on Recommendation Systems*. AAAI Press, Menlo Park California, 1998.

[Valiant, 1979] L. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

[Vontobel, 2010] P. Vontobel. The Bethe permanent of a non-negative matrix. In *Proc. 48th Allerton Conf. on Communications, Control, and Computing*, 2010.

[Wainwright and Jordan, 2008] M. Wainwright and M. Jordan. *Graphical Models, Exponential Families, and Variational Inference*. Now Publishers Inc., Hanover, MA, USA, 2008.

[Wainwright *et al.*, 2002] M. Wainwright, T. Jaakkola, and A. Willsky. A new class of upper bounds on the log partition function. In A. Darwiche and N. Friedman, editors, *Proceedings of the 18th Conference in Uncertainty in Artificial Intelligence*, pages 536–543. Morgan Kaufmann, 2002.

[Wang *et al.*, 2008] J. Wang, T. Jebara, and S. Chang. Graph transduction via alternating minimization. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 1144–1151, New York, NY, USA, 2008. ACM.

[Weimer *et al.*, 2007] M. Weimer, A. Karatzoglou, Q. Le, and A. Smola. COFI RANK - maximum margin matrix factorization for collaborative ranking. In *Advances in Neural Information Processing Systems 20*, 2007.

[Weinberger and Saul, 2009] K. Weinberger and L. Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10:207–244, 2009.

[Weiss, 2000] Y. Weiss. Correctness of local probability propagation in graphical models with loops. *Neural Computation*, 12(1):1–41, 2000.

[Xing *et al.*, 2010] E. Xing, W. Fu, and L. Song. A state-space mixed membership blockmodel for dynamic network tomograph. *Annals of Applied Statistics*, 4(2):535–566, 2010.

[Yedidia *et al.*, 2005] J. Yedidia, W. Freeman, and Y. Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7), 2005.

[Yuille, 2002] A. Yuille. Cccp algorithms to minimize the Bethe and Kikuchi free energies: Convergent alternatives to belief propagation. *Neural Computation*, 14(7):1691–1722, 2002.

[Zhou *et al.*, 2004] D. Zhou, O. Bousquet, T. Navin Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[Zhu *et al.*, 1997] C. Zhu, R. Byrd, P. Lu, and J. Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound constrained optimization. *ACM Transactions on Mathematical Software*, 23(4):550–560, 1997.

[Zhu *et al.*, 2003] X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *International Conference on Machine Learning*, pages 912–919, 2003.

[Ziegler *et al.*, 2005] C. Ziegler, S. McNee, J. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In A. Ellis and T. Hagino, editors, *WWW*. ACM, 2005.