9-2021

Solving Multiple Inference in Graphical Models

Cong Chen
The Graduate Center, City University of New York

SOLVING MULTIPLE INFERENCE IN GRAPHICAL MODELS

by

CONG CHEN

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

2021

Solving Multiple Inference in Graphical Models

by

Cong Chen

This manuscript has been read and accepted by the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

8/6/2021	**Changhe Yuan**
Date	Chair of Examining Committee
8/6/2021	**Ping Ji**
Date	Executive Officer

Supervisory Committee:

Chao Chen

Jia Xu

Neng-Fa Zhou

THE CITY UNIVERSITY OF NEW YORK

ABSTRACT

Solving Multiple Inference in Graphical Models

by

Cong Chen

Advisor: Changhe Yuan

For inference problems in graphical models, much effort has been directed at algorithms for obtaining one single optimal prediction. In practice, the data is often noisy or incomplete, which makes one single optimal solution unreliable. To address this problem, multiple Inference is proposed to find several best solutions, M-Best, where multiple hypotheses are preferred for advanced reasoning. People use oracle accuracy as an evaluation criterion expecting one of the solutions has high accuracy with the ground truth. It has been shown that it is beneficial for the top solutions to be diverse. Approaches for solving diverse multiple inference are proposed such as Diverse M-Best and M-Modes. They rely on hyper-parameters in enforcing diversity. Works keep optimizing the efficiency of solving difficult M-Modes problems by using an intelligent heuristic search on tree decompositions. The newest Min-Loss M-Best introduces a parameter-free method that directly minimizes the expected loss to simultaneously find the multiple top solution set.

Keywords: Graphical Model; Multiple Inference; Oracle Accuracy; M-Best; Diverse M-Best; M-Modes; Min-Lost M-Best; Heuristic Search; Tree Decomposition.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A graphical model in the context is a discrete factor model that represents a set of random variables and their relations via a graph. It represents, explains, and manipulates joint probability distributions. They are commonly used in Bayesian statistics, artificial intelligence, and machine learning (Wainwright and Jordan, 2008). In which case when the graph is undirected, the models are often referred to as Markov random fields, while directed models are Bayesian networks. A graphical model has both a structural or syntax component and a parametric or semantics component. The structural component, encoded by the pattern of edges in the graph. And the parametric component is encoded by numerical potentials associated with sets of edges in the graph.

MAP Inference

For inference problems, much effort has been directed at algorithms for obtaining one single optimal prediction. Maximum a posteriori (MAP) probabilities is one of the basic tasks of graphical model inference, which computes the highest probability configuration over the probability distribution. there are many ways to obtain the highest probability (MAP)

configuration, and various algorithms have been developed for it, including belief propagation (Wainwright, Jaakkola, and Willsky, 2005), tree decomposition (Robertson and Seymour, 1984; Huang and Darwiche, 1996), graph cuts (Boykov, Veksler, and Zabih, 2001; Kolmogorov and Zabin, 2004), integer linear programming (Wainwright, Jaakkola, and Willsky, 2005; Wainwright and Jordan, 2008; Nowozin and Lampert, 2010), and heuristic search (Flerova, Marinescu, and Dechter, 2016).

Multiple Inference

In reality, however, the data are sometimes noisy or incomplete, which makes it necessary to increase the confidence in the answer via finding several best solutions where multiple hypotheses are preferred for advanced reasoning.

Optimization error Bousquet and Bottou (2008); Meltzer, Yanover, and Weiss (2005); Szeliski et al. (2008) is only one aspect of generalization error of learning problems. An exact solution could still be different from the ground truth. The source of this discrepancy may be approximation error, due to the limitations of the models, or estimation error, due to the insufficiencies of the data. For example, in practice, the data is often noisy or missing, which makes the single optimal solution unreliable. So that, this approximation-estimation-optimization tradeoff leads to a more complicated compromise Bousquet and Bottou (2008). In addition to seeking better models, and beyond optimizing more accurately, we would rather look for a set of highly probable solutions than obtaining a single optimal solution, then choose the final solution with expert help or by obtaining additional data.

Therefore, people propose the multiple inference or multiple-prediction problem that aims to find not one, but a set of top candidates for a prediction problem.

It is hard to tell which prediction is better than others from multiple results, thus,

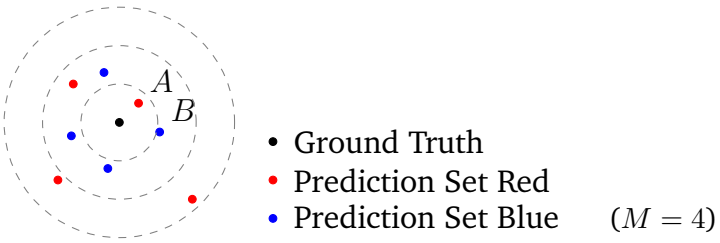
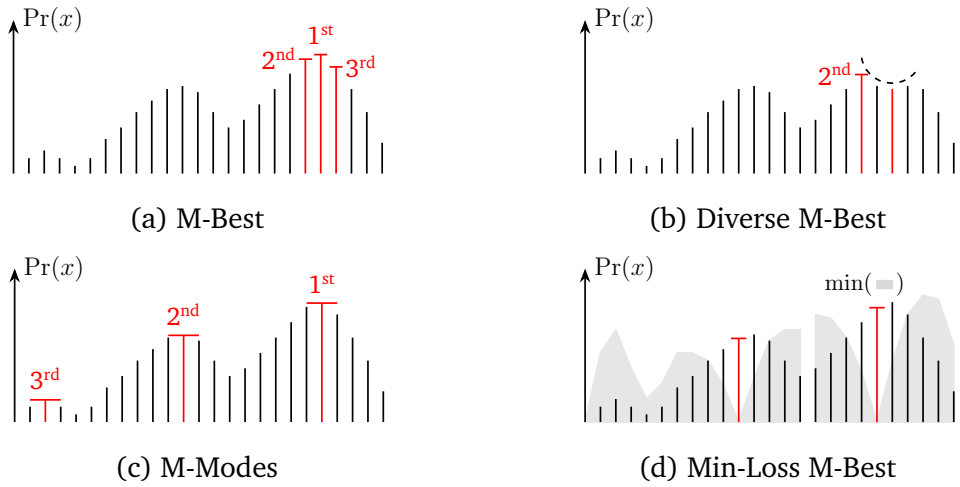


Figure 1.1: Illustration of the concept of oracle accuracy. The central black node represents the ground truth. The red and blue nodes represent two sets of predictions. The best prediction (A) of red set is closer to the ground truth than the best prediction (B) of the blue set, in spite that the blue set is generally all closer to the ground truth. Therefore, the oracle accuracy of set red is higher than the oracle accuracy of set blue.

ideally, after all, we expect one of the given best solution candidates would be chosen as the final answer, i.e. the chosen solution should have very high accuracy. *Oracle accuracy* is used as an evaluation criterion which is defined as the highest accuracy of one of the predictions compared to the ground truth. See Figure 1.1. The problem becomes finding a set of solutions that has the highest oracle accuracy.

To solve the problem, people have proposed different methods targeting better results. M-Best inference (Dechter, Flerova, and Marinescu, 2012) (Figure 1.2(a)) obtains the M most probable predictions. Diverse inference tries to find a set of solutions with both high probability and high diversity. Existing methods, such as diverse M-Best (Batra et al., 2012) (Figure 1.2(b)), which iteratively finds M dissimilar high probable solutions, and M-Modes (Chen et al., 2013) (Figure 1.2(c)), which computes the top M local optima in respective their neighborhoods, rely on a hyper-parameter in enforcing diversity.

Attempts for solving M-Modes (Chen, Yuan, and Chen, 2016; Chen et al., 2014, 2013) demonstrate that the M-Modes is an extremely challenging problem even for a tree model. Dynamic programming is the first algorithm proposed for solving M-Modes in chain and tree graphs (Chen et al., 2014, 2013). Then, a heuristic search approach has been devel-



(a) M-Best (b) Diverse M-Best

(c) M-Modes (d) Min-Loss M-Best

Figure 1.2: Illustration of four multiple inference methods. Each vertical bar corresponds to a labeling while the red bars represent the predictions of each method. The height of the bar corresponds to the labeling's probability. (a) M-Best: it uses M most probable solutions as predictions; (b) Diverse M-Best: it shows the second solution is from the diversity suppression of the first solution; (c) M-Modes: it represents the solutions are top local optimal solutions where neighborhood size $\delta = 1$; (d) Min-Loss M-Best: it shows the two solutions are selected when the whole expected loss (the gray shades) are minimum.

oped to improve the efficiency of solving M-Modes (Chen, Yuan, and Chen, 2016). Later, Chen et al. (2018) presents a more general implementation of the heuristic search method based on tree decompositions. Recently, Chen, Yuan, and Chen (2020) measures and optimized search orderings, which were shown to result in up to orders of magnitude speedups in the experiments.

These approaches for solving multiple inference are all derivatives of finding the posterior mode(s), aka the MAP estimates. They climb the probability distribution landscape: Routed from the top (MAP), then either choose the top one or jump to farther ones preventing similarity. The MAP estimation targets to find the high probable point(s) without taking the volume of the landscape into account.

However, it is important to point out that MAP estimation has drawbacks. The mode(s) are usually quite untypical of the distribution (Murphy, 2012). Different from MAP estimation, Bayesian methods are characterized by the use of distributions to summarize data and draw inferences. In contrast to MAP-based inference, the new objective remodels optimizing oracle accuracy and directly minimizes expected loss in finding high-accuracy multiple solutions. Min-Loss M-Best(Chen et al., 2020) (Figure 1.2(d)) aims to jointly find M solutions, which at least one of them has the lowest expected loss.

In this work, we study the evolution of different published approaches for solving multiple inference in graphical models. We begin with the background of graphical models, MAP inference, and multiple inference. Multiple inference evolves from M-Best, Diverse M-Best, to M-Modes. We study an improvement of the current heuristic search method for M-Modes by optimizing a better subgraph ordering. Last, we developed a fundamental formulation for multiple inference by directly optimizing oracle accuracy via expected loss.

Chapter 2

Background

2.1 Graphical Models

We begin with the background on graphical models. We start by formally defining the graphical models, introduce the two most important sub-model Bayesian networks and Markov random field, and discuss their reasoning task—inference algorithms.

The key idea is that of factorization: A *graphical model* is a collection of factor functions over subsets of variables that conveys probabilistic, deterministic, or preferential information. The structure can be visualized as a graph for understanding convenience. The graph captures independencies or irrelevance information inherent in the model, that can be useful for interpreting the modeled data and exploited by reasoning tasks. Wainwright and Jordan (2008); Flerova, Marinescu, and Dechter (2016)

OpenGM 2.0 Manual (Andres, Beier, and Kappes, 2012) (a template library) gives a rigorous definition of graphical models. It defines the mathematical foundation from the syntax which determines the conditional independence assumptions, to the semantics specify the operations consistent with the syntax. We simplified this definition (eliminating

the concept of function identifiers) as follows:

The *syntax* consists of a factor graph (V, X, F), which V are the discrete random variables, X are these variables domains, and F are factor functions which depends on these variables.

Random variables, $x_{v_1}, x_{v_2}, \ldots x_v$, correspond to vertices, $v \in V$, in the graph. The terms of variable and vertex are used interchangeably. The variable can be assigned a discrete value, or a label, of which its own set of their respective finite domains, e.g., $x_v \in \{ d_1, d_2, \ldots, d_v \}$. A vector, or an ordered subset of variables forms an configuration, or labeling, $x = (x_1, x_2, \ldots, x_n)$.

Factor functions, $f \in F$, are functions over a vector of variables. $\mathcal{N}(f)$ denotes the subset of variables on which f applies. Factor functions can also be called potential functions or potentials.

With respect to a given syntax, the *semantics* consist of one finite set $X_v \neq \varnothing$ for each $v \in V$, a commutative monoid $(\Omega, \odot, 1)$ and one function $f \in F$, such that $f : X_{v_1^{(f)}} \times \cdots X_{v_{|\mathcal{N}(f)|}^{(f)}} \to \Omega$. A monoid is an algebraic structure with a single associative binary operation and an identity element.

The function $f : X \to \Omega$ such that $\forall (x_{v_1}, \ldots, x_{v_{|V|}}) \in X$:

$$f(x_{v_1}, \ldots, x_{v_{|V|}}) := \bigodot_{f \in F} f(x_{v_1^{(f)}}, \ldots, x_{v_{|\mathcal{N}(f)|}^{(f)}}) \tag{2.1}$$

For example, a graphical model $M = (V, X, F)$, where $V = \{0, 1\}$, that there are two variables, $X = \{X_1, X_2\}$. $X_1 = \{0, 1\}$, and $X_2 = \{0, 1\}$, that each variable can be assigned either 0 or 1. And

$$F = \{ f_0(x_0 = 0) = 1, f_0(x_0 = 1) = 2, f_1(x_1 = 0) = 3, f_1(x_1 = 1) = 4, \tag{2.2}$$

$$f_{01}(x_0 = 0, x_1 = 0) = 5, f_{01}(x_0 = 0, x_1 = 1) = 6, \tag{2.3}$$

$$f_{01}(x_0 = 1, x_1 = 0) = 7, f_{01}(x_0 = 1, x_1 = 1) = 8 \} \tag{2.4}$$

, that there are unary and binary factor functions over their variables defined, where $\odot = \sum$ and $\Omega = \mathbb{R}$. Therefore we can induce that:

$$f(x_0 = 0, x_1 = 1) = f_0(x_0 = 0) + f_1(x_1 = 1) + f_{01}(x_0 = 0, x_1 = 1) \tag{2.5}$$

$$= 1 + 4 + 6 = 11 \tag{2.6}$$

Bayesian Networks (Directed Graphical Models)

When a graphical model's operator $\odot = \prod$ and $f(v) = \Pr\big(v \mid \text{parent}(v)\big)$, we have a *bayesian network* (BN), so that $\Pr(x) = \prod_{v \in V} \Pr\big(x_v \mid \text{parent}(x_v)\big)$. A Bayesian network (BN) (Pearl, 1988) is a probabilistic graphical model that represents probabilistic dependencies between random variables using a directed acyclic graph (DAG).

Markov Random Fields (Undirected Graphical Models)

When a graphical model's operator $\odot = \sum$ and $f_c(x_c) = -\log\big(\psi(x_c)\big)$, where $\Pr(x) = \frac{1}{Z}\prod_{c \in C} \psi(x_c)$, in which Z is the normalization term and C are the set of all maximal cliques. A *Markov random fields* (MRF) (Wainwright and Jordan, 2008; Nowozin and Lampert, 2010) is a probabilistic graphical model that represents a joint discrete distribution using an undirected graph and potential functions associated to its maximal cliques.

If we denote $f(x) = -\log\big(\prod_{c \in C} \psi(x_c)\big)$ as the energy of configuration x, it can be decomposed into sums over maximal cliques $f(x) = \sum_{c \in C} f_c(x_c)$, in which $f_c(x_c) = -\log\big(\psi(x_c)\big)$. The energy is inversely proportional to the $\log$ of the probability. Therefore, finding a solution with the maximal probability is equivalent to minimizing the energy function.

Constraint Networks

When a graphical model's operator $\odot = \bigwedge$ and $f(x) = \bigwedge_c f_c(x)$, where $f_i(x) = \text{true}$ or false, so that overall function is the *and*s of individual constraints.

2.2 MAP Inference

Given a graphical model and, rather than the commutative monoid $(\Omega, \odot, 1)$, a commutative semi-ring $(\Omega, \odot, 1, \oplus, 0)$, the problem of computing

$$\bigoplus_x F(x) \quad \text{i.e.} \quad \bigoplus_{x \in X} \bigodot_{f \in F} f(x_{v_1^{(f)}}, \ldots, x_{v_{|\mathcal{N}(f)|}^{(f)}}) \tag{2.7}$$

is a central problem in machine learning. A ring consists of a set equipped with two binary operations that generalize the arithmetic operations of addition and multiplication. A semi-ring is similar to a ring, but without the requirement that each element must have an additive inverse. It can have instances in 1) marginalization $(\mathbb{R}^+, \cdot, 1, +, 0)$:

$$\sum_{x \in X} \prod_{f \in F} f(x_{v_1^{(f)}}, \ldots, x_{v_{|\mathcal{N}(f)|}^{(f)}}) \tag{2.8}$$

2) optimization $(\mathbb{R}, +, 0, \min, \infty)$:

$$\arg\min_{x \in X} \sum_{f \in F} f(x_{v_1^{(f)}}, \ldots, x_{v_{|\mathcal{N}(f)|}^{(f)}}) \tag{2.9}$$

and 3) constrained satisfaction $(0, 1, \wedge, 1, \vee, 0)$:

$$\bigvee_{x \in X} \bigwedge_{f \in F} f(x_{v_1^{(f)}}, \ldots, x_{v_{|\mathcal{N}(f)|}^{(f)}}) \tag{2.10}$$

We focus on optimization, i.e. $\arg\min_x F(x)$.

The most common optimization task for Bayesian networks or Markov random fields is maximum a posteriori (MAP) probabilities, also known as the most probable explanation

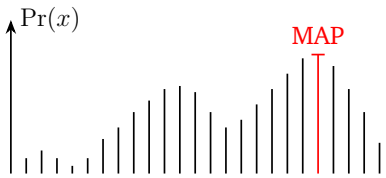


Figure 2.1: Maximum a posteriori (MAP)

(MPE), that computes the highest probability configuration over the probability distribution. The highest configuration is the global optimum of the probability landscape. It is the same as minimizing the potential function of a given graphical model. See Figure 2.1. There are many algorithms to solve MAP problems that are not limited to such as, Belief Propagation, Tree Decomposition, Graph Cut, Integer Linear Programming, and Heuristic Search.

$$y^* = \arg\max_{y \in \mathcal{Y}} \Pr(y) = \arg\min_{y} \mathrm{E}(y) \tag{2.11}$$

Belief Propagation

Belief propagation (Pearl, 1982; Wainwright and Jordan, 2008) (BP), also known as sum-product message passing, is a message-passing algorithm for performing inference on graphical models. It calculates the marginal distribution for each unobserved node, conditional on any observed nodes. It is an exact inference algorithm on trees, while it is not exact on general loopy graphs.

Roughly speaking, the algorithm sums over branches at the vertices, and performs minimizations at the edges; The minimizations take place when vertices' potentials are being propagated to their neighboring edges. This algorithm is crucial and fundamental for MAP calculations.

Tree Decomposition

If we apply BP on general loopy graphs, a typical way is to decompose the graph into a junction tree. A *tree decomposition* (Robertson and Seymour, 1984; Huang and Darwiche, 1996; Lauritzen and Spiegelhalter, 1988) is a mapping of a graph into an undirected tree, called *junction tree*. A junction tree consists of *clusters* and *sepsets*. Each vertex in a junction tree is a cluster of variables; A sepset connecting two adjacent clusters represents the intersection of the clusters.

The junction tree encodes a joint probability distribution of the labeling for the graph according to:

$$\Pr(x) = \frac{\prod_i \Pr(C_i)}{\prod_j \Pr(S_j)} \tag{2.12}$$

where $\Pr(C_i)$ and $\Pr(S_j)$ are joint probability distributions for clusters C_i and sepsets S_j respectively. And for each cluster C and neighboring sepset S, it holds that $\Pr(S) = \sum_{C \setminus S} \Pr(C)$.

Graph Cut

Boykov, Veksler, and Zabih (2001); Kolmogorov and Zabin (2004) introduce an idea of using max-flow and min-cut to solve MAP problem on loopy graphical models, called *Graph Cut*. This clever idea makes NP solution into P solution. Further, it introduces α-expansion algorithm which even make similar algorithm can solve larger label size models. One problem of this is that the model potentials should accept a certain property which sum of the diagonal values should be greater or equal to the others, which makes this algorithm be constricted, similar to the binary scenarios: $f(01) + f(10) \geqslant f(00) + f(11)$ where f here are binary potentials.

Integer Linear Programming

MAP problem can formulate into *Integer Linear Programming* (ILP) problem (Wainwright, Jaakkola, and Willsky, 2005; Wainwright and Jordan, 2008; Nowozin and Lampert, 2010) such that makes the constraints over the factors and minimize the problem to get the solution. This method could also give approximate solutions via relax integrality constraints as linear programming (LP).

The MAP problem can be formulated as an ILP as follows:

$$\mathrm{MAP}(\theta) = \min_{\mu} \left[\sum_{i \in V} \sum_{x_i} \theta_i(x_i)\mu_i(x_i) + \sum_{ij \in E} \sum_{x_i, x_j} \theta_{ij}(x_i, x_j)\mu_{ij}(x_i, x_j) \right] \tag{2.13}$$

, subject to:

- $\mu_i(x_i) \in \{ 0, 1 \} \quad (\forall i \in V, x_i)$
- $\sum_{x_i} \mu_i(x_i) = 1 \quad (\forall i \in V)$
- $\mu_i(x_i) = \sum_{x_j} \mu_{ij}(x_i, x_j) \quad (\forall ij \in E, x_i)$
- $\mu_i(x_j) = \sum_{x_i} \mu_{ij}(x_i, x_j) \quad (\forall ij \in E, x_j)$

MAP inference as a discrete optimization problem is to identify a configuration with minimizing the summation of total potentials. Each potential function is assigned a μ_f to indicate whether choose the function or not. The first constraint is the integrality constraints that decide the choices. Relaxing integrality constraints allow variables to be between 0 and 1. The second constraint force every "cluster" of variables to choose a local assignment. The third constraint enforce that these assignments are consistent.

LP can be solved efficiently, such as simplex, interior point, and ellipsoid algorithm. Since the LP relaxation maximizes over a larger set, its value can only be lower. For integer linear programs, we can use local search, branch-and-bound, and branch-and-cut methods to solve.

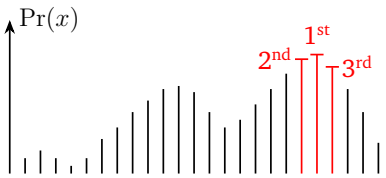


Figure 2.2: M-Best

Heuristic Search

Search algorithms provide a way to systematically enumerate all possible configurations of a given graphical model. Optimization problems over graphical models can be naturally presented as the task of finding a shortest path in an appropriate search space.

Using an OR search tree is straightforward. Each tree level corresponds to a variable. The states correspond to partial variable configurations and the arc costs come from the model's potential functions.

OR search trees are blind to the problem decomposition encoded in graphical models and can therefore be inefficient. They do not exploit the independencies in the model. AND/OR search spaces have been introduced to better capture and exploit the features. (Dechter and Mateescu, 2007)

2.3 M-Best Multiple Inference

Finding an optimal solution is always the usual aim of graphical model inference problems. However, in reality, the data is sometimes noisy or uncompleted, which makes obtaining a single optimal solution questionable. It is possible to increase the confidence of the answer via finding several best solutions and then let an expert choose a final solution or via obtaining additional data or ranked using a secondary mechanism (Li, Carreira, and

Sminchisescu, 2010). Thus, there needs to acquire more than one optimum, but a set of M best solutions. See Figure 2.2.

More work of the M-Best is for procurement auction problems and probabilistic expert systems. Either because certain constraints make the problem infeasibly complex or they are too vague to formalize, they cannot be directly incorporated within the model. So that it may be more practical to separate into two stages: first find several good solutions to a relaxed problem, and then pick the one that meets all additional constraints as a post-processing second step. In addition, in the context of summation problems over graphical models, such as model averaging, an approximation can be derived by summing over the M most likely tuples (Tian, He, and Ram, 2012; Chen and Tian, 2014).

k Shortest Path (Nillson's Approach)

The k shortest path routing algorithm is a generalized algorithm of the shortest path routing algorithm in a given network, not only finds the shortest path, but also other paths in ascending order of cost. Since (Hoffman and Pavley, 1959), there have been many papers published on the k shortest path routing algorithm problem. If not considering the loop, yen's algorithm (Hoffman and Pavley, 1959; Yen, 1971) is the most classic one. The idea is to break down the search iteratively into two parts, 1) determining the first k-shortest path, and then 2) determining all other k-shortest paths via detouring each node.

Lawler (Lawler, 1972), one of the earliest and most influential works, proposed a general algorithm to compute the top M solutions for a large family of discrete optimization problems via using similar ideas from k shortest path. The idea is to compute the next best solution successively by finding a single optimal solution for a slightly different reformulation of the original problem that excludes the solutions generated so far. This approach is still one of the primary strategies for finding the M best solutions. Other approaches are more direct, trying to avoid the repeated computation inherent to Lawler's scheme. Most

relevant work is by Nilsson (Nilsson, 1998).

Nilsson's Approach, which applied Lawler's method, proposed a junction-tree-based message-passing scheme that iteratively finds the M best solutions. He uses message-passing on a junction tree to calculate the initial max-marginal functions for each cluster yielding the best solution. Subsequent solutions are recovered by conditioning search which consults the generated function. There are three phases to find M-Best configurations: 1) partition-phase, 2) candidate-phase, and 3) identification-phase. Via utilizing the junction tree property, they present a cleverer partitioning scheme preventing the useless calculating of irrelevant subsets or clusters.

Heuristic Search (Dechter's Approach)

The contribution of Dechter's approach lies in extending the heuristic search to the M best solutions task. They propose general-purpose M-best variants of both best-first search, which is that A* yields M-A*, and bound depth-first search, which is that DFBnB yields M-BB, respectively. They show that M-A* inherits all A*s desirable properties (Dechter and Pearl, 1985), most significantly it is optimally efficient compared to any alternative exact search-based scheme. They also extend their new M-best algorithms to graphical models by exploring the AND/OR search space.

M-A* and M-BB: The M-best tree-search variant of A*, denoted *M-A**, solves an M-Best optimization problem over any general search graph. The scheme expands the nodes in the order of increasing value of f in the usual A* manner. M-A* picks the node with the smallest evaluation function $f(n)$ in the open list and puts it in the closed list. If a state is a goal, a new solution is reported. Otherwise, the state is expanded and its children are created. States encountered beyond M times are discarded.

Algorithm *M-BB*, the depth-first branch and bound extension to the M-Best task. Other

than similar setups, it also maintains a keeping sorted list of candidate states that contains the best M solutions found so far. The open list is organized as a stack in order to facilitate a depth-first exploration. If a state is a goal node, a new complete solution is found and it is stored in the candidate list. Only up to M best solutions are maintained. Before M solutions are discovered, no pruning takes place. Then the worst candidate state would be the upper bound used for pruning. When the algorithm terminates, it outputs the M Best solutions to the problem.

AND/OR Search Space: The AND/OR search space captures models' decomposition. AND/OR search tree contains alternating levels of OR and AND nodes. Merging all context-mergeable nodes yields the context-minimal AND/OR search graph. The size of the context-minimal AND/OR search graph is exponential to the induced width, denoted as w^*, of the original graph. In the tree decomposition context, induced width is also called tree-width, which is the largest size - 1 of the cluster via the optimal tree decomposition. The AND/OR version of M-A* is called *M-AOBF*, and AND/OR version of M-BB is *M-AOBB*.

MBE Heuristics: Their search methods often use the Mini-Bucket Elimination (MBE) as heuristic functions (Dechter and Rish, 2003), which is an approximate version of an exact variable elimination algorithm called Bucket Elimination (BE) (Dechter, 1999). Given a variable ordering, the algorithm associates each variable with a bucket containing all potentials defined on this variable. Large buckets are partitioned into smaller subsets, called mini-buckets. The parameter i, the size of the mini-buckets, is called the i-bound. MBE generates a lower bound on the optimal value. Higher i values take more computational resources but yield more accurate bounds. When bucket size is large enough, that MBE becomes BE, with best first search scheme, we get *BE+M-BF* algorithm.

Another approach (Flerova, Rollon, and Dechter, 2012) extends bucket elimination and mini-bucket elimination to the M-best solutions task, yielding an exact scheme called *Elim-M-Opt*.

Other Approach (ILP)

One of the popular approximate approaches to solving optimization problems is based on the LP-relaxation of the problem (Yanover and Weiss, 2004). The m-best extension of this approach does not guarantee exact solutions but is quite efficient in practice. However, on trees or junction-trees, it is exact if the underlying LP solver reports solutions within the time limit.

To construct an LP whose solution is second best, a natural approach is to use the LP for (first best) MAP, and then somehow eliminate the solution using additional constraints. The constraint is that $\exists \mu_i(x_i), x_i \in x^*$ s.t. $\mu_i(x_i) = 0$.

Time Complexity Comparison

These are the time complexity comparison of the exact M-best algorithms aforementioned specified for a bucket tree (Flerova, Marinescu, and Dechter, 2016). Problem parameters: n – number of variables, k – largest domain size, w^* – induced width, $\deg$ – the degree of the join (bucket) tree.

- **Lawler:** $\mathcal{O}\big(\, n^2 \cdot m \cdot k^{w^*} \,\big)$
- **Nilsson:** $\mathcal{O}\big(\, 2nk^{w^*+1} + mn(2\log(mn) + 2k) \,\big)$
- **M-A*-tree:** $\mathcal{O}\big(\, k^n \,\big)$
- **M-BB-tree:** $\mathcal{O}\big(\, k^n + \log M \,\big)$
- **M-AOBF:** $\mathcal{O}\big(\, n \cdot k^{w^* \log n} \,\big)$
- **M-AOBB:** $\mathcal{O}\big(\, m \cdot n \cdot k^{w^*} \cdot \deg \cdot \log m \,\big)$
- **BE+M-BF:** $\mathcal{O}\big(\, n \cdot k^{w^*+1} + mn \,\big)$

2.4 Diverse M-Best Multiple Inference

In contrast to the M-Best MAP problem that involves finding the top M most probable solutions, diverse inference highlights the concept *diversity*, leads the first approach Diverse M-Best (Batra et al., 2012). This approach seeks to produce highly probable solutions that are qualitatively different not only 1) from the MAP but also 2) from each other. This is an important distinction because the literal definition of M-best MAP is not expected to work well in practice: any reasonable setting of M would return solutions nearly identical to the MAP when the problem is in a large state-space problem.

Such a diverse solution set is of interest in computer vision (Yadollahpour, Batra, and Shakhnarovich, 2013; Kirillov et al., 2015a) and computational biology (Fromer and Yanover, 2009) where multiple hypotheses are preferred for advanced reasoning.

Recent empirical studies (Meltzer, Yanover, and Weiss, 2005; Szeliski et al., 2008; Ladicky et al., 2010) keep finding existing models' MAP solutions are having much poorer quality than the ground truth on vision problems like segmentation, stereo, optical flow, denoising, etc. That is to say, the ground truth has a lower probability than the MAP solution under existing models. This discrepancy has been the driving force demanding a better model for optimization.

Distance Measures

A dissimilarity function $\Delta(\,\{x\}\,)$ (a shortcut for $\Delta(x_1, x_2, \dots)$) is used to define the distance between several labelings, i.e. $\Delta(\,\cdot\,)$ takes a large value if elements in $\{x\}$ are different, and a small value otherwise. This distance measure can be classified into nodewise and pairwise distances (Kirillov et al., 2015b).

Nodewise distance: $\Delta(\,\{x\}\,) = \sum_v \Delta_{[v]}(\,\{x_{[v]}\}\,)$

Pairwise distance: $\Delta(\,\{x\}\,) = \sum_{i,j} \Delta(x_i, x_j)$

Hamming distance: $\Delta(\,\{x\}\,) = \sum_{i,j,v} [\![\, x_{i[v]} \neq x_{j[v]} \,]\!]$

Hamming distance is a special case of both nodewise and pairwise distances, besides an indicator function for each disagreed variable value.

Oracle Accuracy

As commonly done in the multiple prediction literature (Batra et al., 2012; Gimpel et al., 2013; Chen et al., 2018), *oracle accuracy* is often used as the evaluation criterion in multiple predictions, i.e., the highest accuracy of one of the M predictions compared to the ground truth, y_{gt}. In the context of multiple inference tasks, when the model has already correctly depicted the real distribution, a ground truth labeling can be regarded as a sample drawn from the model distribution, i.e., $y_{\text{gt}} \sim \mathcal{Y}$. Oracle accuracy doesn't care about whether the remaining solutions are poor quality or not, but one solution with the highest accuracy is accepted. The oracle accuracy of a set of M labeling, $\{y\}_M$, by Hamming distance, can be calculated as:

$$\mathsf{OrcAcc}\big(\{y\}_M\big) = \frac{|y| - \min\big(\Delta(y_1, y_{\text{gt}}), \ldots, \Delta(y_M, y_{\text{gt}})\big)}{|y|} \tag{2.14}$$

Here, the $|y|$ represents the variable size of y. Often, we also use *error rate* instead of accuracy which is $\mathsf{ErrRate}(y) = 100\% - \mathsf{OrcAcc}(y)$

Diverse M-Best

Batra et al. (2012); Prasad, Jegelka, and Batra (2014) proposed a method that is more efficient to generate M diversified solutions. It works iteratively, starting with the MAP, which is the first solution, but it iteratively finds the next solution via a regularization of

a diversity penalty term that it has to be dissimilar by a certain margin from the solutions collected so far. This method is very efficient and easy to implement.

Formulation: We now describe the proposed Diverse M-Best formulation. Recall that the goal is to produce a diverse set of low-energy solutions. We approach this problem with a greedy algorithm, where the next solution is defined as the lowest energy state with at least some minimum dissimilarity from the previously chosen solutions. To do so, we assume access to a dissimilarity function $\Delta(y^a, y^b)$ between solutions which defines the diversity of two configurations. Let y^m denote the m$^{\text{th}}$-best solution. Thus y^1 is the MAP, y^2 is the second-best, and so on. They propose the following straightforward yet fairly general formulation:

$$y^* = y_1 = \arg\min_{y \in \mathcal{Y}} \Big[\mathrm{E}(y) \Big] \tag{2.15}$$

$$y_2 = \arg\min_{y} \Big[\mathrm{E}(y) - \lambda \Delta(y, y_1) \Big] \tag{2.16}$$

$$\vdots \tag{2.17}$$

$$y_m = \arg\min_{y} \Big[\mathrm{E}(y) - \lambda \sum_{i=1}^{m-1} \Delta(y, y_i) \Big] \tag{2.18}$$

And see Figure 2.3. The λ coefficient measures how strong the penalty term is. It could be decided by cross-validation.

M-Best algorithm is a special case of Diverse M-Best. It is defined as when $\Delta(x^a, x^b) = [\![\, y^a \neq y^b \,]\!]$ and $\lambda = 1$, where $[\![\, \cdot \,]\!]$ is an indicator function. Thus forces the next best solution not to be same as MAP.

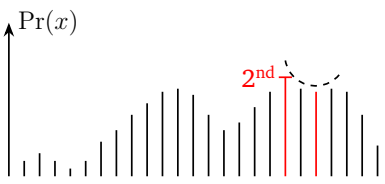


Figure 2.3: Diverse M-Best

Remarks

Diverse M-Best formulation is the first approach that proposes an elegant way to trade-off between optimization and diversity. One issue of practical concern is how the parameter λ is chosen. This is related to the topology of the distribution landscape. If λ is set too low, the next solution may still be trapped at the same peak. If λ is set too high, then several solutions will be ignored. An appropriate value of λ is problem-dependent and must be tuned as a free parameter. So, tuning the value of λ is similar to a regularizer in learning. In practice, the regularizer is tuned via cross-validation.

The Problem of Non-discrete λ: Diverse M-Best has a critical problem that it needs toning penalty terms λ, normally via cross-validation. Such a term is a continuous numeric that it is difficult to find an exact good solution for all problems. And it is even different in each case. So people introduce a better measure that is other than the complicated term but by a more discrete method—M-Modes.

Chapter 3

M-Modes Multiple Inference

3.1 M-Modes

The *M-Modes* (Chen et al., 2013) method is the problem of computing the top M locally optimal configurations, each of which has higher quality than all other solutions within a given scalar distance δ. These locally optimal solutions, called *modes*, capture the topographical features of the probabilistic landscape of the given graphical model, and are also highly possible and are naturally diverse. See Figure 3.1.

Given a non-negative integer δ, δ-neighborhood $\mathcal{N}_\delta(y)$ is defined as $\mathcal{N}_\delta(y) = \{\, x \mid \Delta(x, y) \leqslant \delta \,\}$. So, a labeling y is a δ-mode as y has highest probability (lowest energy)

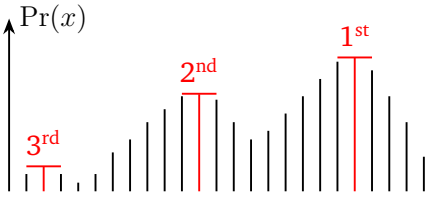


Figure 3.1: M-Modes

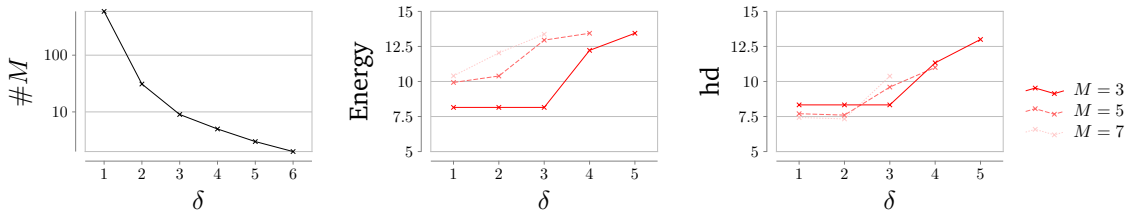


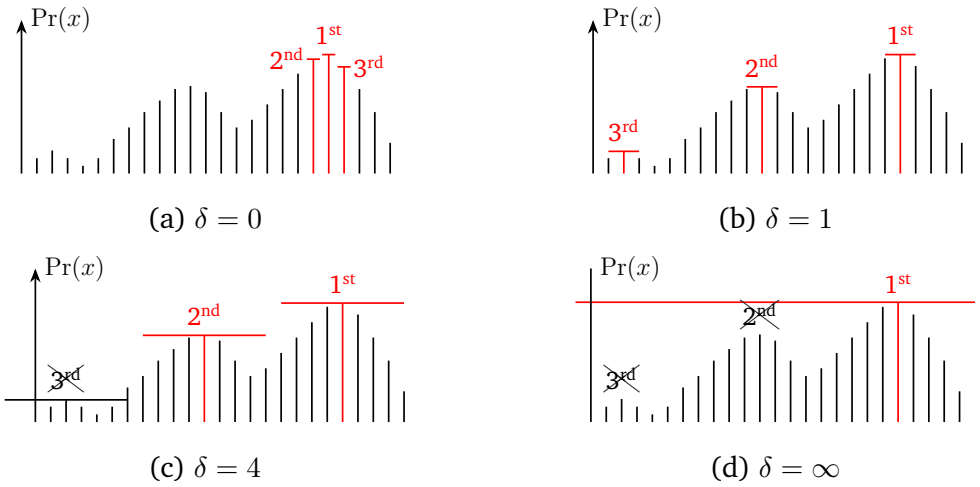
Figure 3.2: The effect of δ on (Left) total number of modes, (Middle) the energy of the M-th mode (negatively proportional to log probability), and (Right) the average pairwise hamming distance on dataset Child.

in its δ-neighborhood. For simplicity, we drop δ from the notations hereafter. Therefore, M-Modes is a problem to compute the top M best modes.

The concept of δ-neighborhood ensures the modes are diverse; any two modes are at least δ away. But if δ is too large, too many high-probability solutions are suppressed by superior neighbors, and the top modes may contain too many low-probability solutions. Therefore, the δ provides a tradeoff between the diversity and probability of modes. Figure 3.2 provides visual illustrations of the effect of δ on a UCI dataset called Child. And Figure 3.3 illustrates what mode solutions and δ-neighborhoods are and how the scale δ affects the number of modes. When $\delta = 0$, so every labeling is a mode so that the problem becomes M-Best. When $\delta = \infty$, the MAP solution is the only mode.

3.1.1 Global-Local Theorem

To compute M-Modes, one has to leverage the relationship between a mode and its local patterns. Given a graph G, we are particularly interested in its connected subgraphs with size δ, called δ-*subgraphs*. For a δ-subgraph, S_δ or S for simplicity, all variables that are adjacent to variables in S are its *boundary* or *boundary variables*, denoted as ∂S. Denote by $cl(S)$ the disjoint union of S and its boundary ∂S, $S \sqcup \partial S$. For convenience, we also call the variables of S *interior variables*. Figure 3.4(a) shows a δ-subgraph with $\delta = 3$ in a grid



(a) $\delta = 0$ (b) $\delta = 1$

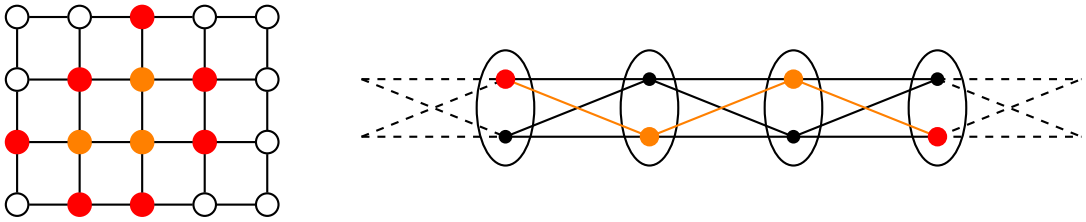
(c) $\delta = 4$ (d) $\delta = \infty$

Figure 3.3: An illustration of modes under different δ. Each vertical bar corresponds to a labeling, and the height corresponds to its probability. (a) When $\delta = 0$, every labeling is a mode, and M-Modes reduces to M-Best. (b) When $\delta = 1$, there are three modes. (c) When $\delta = 4$, only two modes are left. The third mode is no longer locally optimal in its δ-neighborhood. (d)When $\delta = \infty$, only the first mode is a mode, and M-Modes reduces to MAP.

graph.

A label assignment to a δ-subgraph, S, is called a *local labeling*, l_δ or l. Given a label assignment to boundaries of S, $l \in D^{\partial S}$, the highest precedential local labeling of S is called a *local MAP*, $\hat{l}$. Consider Figure 3.4(b), fixing the labels of the boundary (red) can uniquely determine the local MAP of interior variables (orange). We say two local labelings are consistent if they agree on the overlaps. For example,$\langle 101 \dots \rangle$ and $\langle \_011 \dots \rangle$ are consistant for they both have same label at overlaps, variable 2 and 3.

It was shown that there is a close connection between the modes of a graph and the local MAPs of the δ-subgraphs. In particular, any consistent combinations of local MAPs is a global mode, and vice versa (Chen et al., 2014). This property has been exploited by several recent algorithms for solving M-Modes (Chen et al., 2014; Chen, Yuan, and Chen,



(a) $\delta = 3$ (b) $\delta = 2$

Figure 3.4: Examples of δ-subgraphs. (a) The red vertices forms the boundary contain three orange interior variables on a grid graph. (b) A trellis diagram represents a chain. The ellipses are variables, and the dots are its two labels. Where the boundary variables (red) are fixed, it can exactly solve its local MAP (orange).

2016). Formally, we have:

Theorem 1 (Global-Local)**.** *A labeling is a δ-mode if and only if its local labelings of all δ-subgraphs are local MAPs.*

3.1.2 δ-subgraphs

To use the Global-Local Theorem to help us finding M Modes from a given model, we should explore all δ-subgraphs of a graphical model as a prepocessing step. The calligraphic font $\mathcal{S}_\delta$ or $\mathcal{S}$ denotes the set of all the δ-subgraphs (interiors) of a given graph. We use an algorithm to compile $\mathcal{S}_\delta$ from a given model, G. This algorithm is a dynamic programming work of iterating the found δ-subgraph sets from $\delta = 1, 2$ until δ.

First, for $\delta = 1$ and 2 are trivial, whereas these are sets of all the vertices and the edges of the graph. For each of the 2-subgraphs, we keep adding its neighbors and forms 3-subgraphs, then insert them into the 3-subgraph set. By keep doing this way, we iterate this step until we reach to finish the δ size set. See Algorithm 1 for details. And Table 3.1 shows the number of δ-subgraphs of different δ size (δ-subgraph set sizes) on three benchmark

Algorithm 1 Compiling δ-subgraph Set

Input: G: a graphical model, δ: δ size

Output: $\mathcal{S}_\delta$: a δ-subgraph Set

```
1: function COMPILE-SUBGRAPH-SET(G, δ)
2:     if δ = 1 then return V                          ▷ the set of vertices
3:     else if δ = 2 then return E                     ▷ the set of edges
4:     else
5:         S_{d=2} ← E
6:         repeat
7:             S_{d+1} ← ∅
8:             for all S_d ∈ S_d do
9:                 for all v ∈ ∂S_d do
10:                    S_{d+1}.add(S_d + v)
11:                end for
12:            end for
13:            d ← d + 1
14:        until d = δ
15:        return S_δ
16:    end if
17: end function
```

datasets. It empirically shows δ-subgraph set sizes are about exponentially increasing to δ size ($\delta \ll |V|$).

3.1.3 Solving M-Modes using Dynamic Programming

Solving M-Modes is much more challenging than M-Best. To solve the latter problem, we only need to globally compare the probability of different label configurations. For M-

Table 3.1: δ-subgraph set sizes on dataset Child, Alarm, and Hepar2

$\delta =$	1	2	3	4	5	6	7	8
Child	20	32	92	282	770	1,785	3,498	5,843
Alarm	37	70	216	704	2,237	6,817	19,895	55,836
Hepar2	70	161	1,380	11,939	93,630	658,728	4,171,393	23,971,053

Modes, however, a solution should not only have a high probability but also be the mode in its local neighborhood. Based on this Global-Local Theorem, a dynamic programming algorithm, where Chen et al. (2013) on chains, and Chen et al. (2014) on trees, has been developed to solve M-Modes by first computing all local MAPs of each δ-subgraph and then searching through all their consistent combinations to find the top M ones. See Figure 3.5. Also see Algorithm 2.

The computational bottleneck is the calculation of all the local MAPs for all the δ-subgraphs. The number of local MAPs on each subgraph can be very large as equals to $|D^{\partial S}|$.

3.2 Solving M-Modes using Heuristic Search

Another Chen, Yuan, and Chen (2016) proposes to apply heuristic search to solve M-Modes. This search algorithm also builds on the Global-Local theorem, but directly searches for global modes by incrementally piecing together consistent local MAPs. Such search is enabled by a novel search operator designed to search the local modes of a δ-subgraph of variables at each step. The local MAPs are not computed *a priori*, but are generated and verified *on the fly*.

Finding M-Modes using heuristic search is to apply depth-first branch and bound (DF-

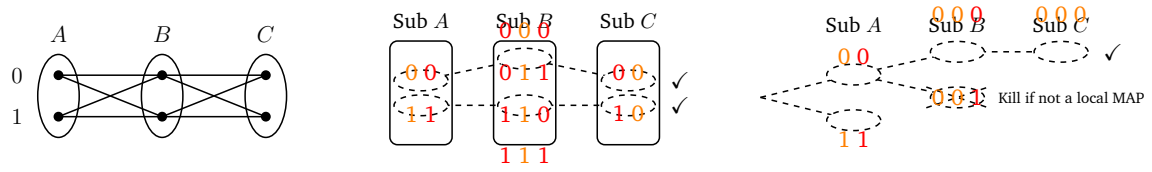
BnB) or A* to search for consistent local MAPs of the δ-subgraphs. The local MAPs are generated and verified when they are needed. An admissible heuristic function is used to evaluate the promisingness of a search MAPs. The function can be obtained by computing the MAP solution over the remaining variables that are yet to be searched. It is calculated only once in the reverse vertex ordering, with which δ-subgraph ordering is coordinated, and stored for repeated queries by the global modes search.

Then, the algorithms use the ordered δ-subgraphs to form layers of search nodes in a search tree. If a δ-subgraph contains new boundary variable(s), each instantiation of the new boundary variables needs to be *searched* so that the conditional local MAP over the interiors is computed or verified, which may lead to a successor node. Otherwise, the δ-subgraph corresponds to a pure *verification* layer without branching. Once all δ-subgraphs are explored, a solution is found. See Figure 3.5(c). The algorithms terminate when either the whole search space is explored (M-DFBnB) or top M solutions have been found (M-A*). See Algorithm 3 for details of M-A*.

3.3 Solving M-Modes using Tree Decompositions

The implementation in Chen, Yuan, and Chen (2016) was tailored for special models such as trees and submodular graphs. Chen et al. (2018) provides a more general implementation of the search method based on tree decompositions that applies to general loopy graphs. See Table 3.2

A *tree decomposition* (Robertson and Seymour, 1984), also called *junction tree*, is a mapping of a graph into an undirected tree. A junction tree consists of *clusters* and *sepsets*. The junction tree encodes a joint probability distribution of the labeling for the graph according to: $\Pr(x) = \frac{\prod_i \Pr(C_i)}{\prod_j \Pr(S_j)}$, where $\Pr(C_i)$ and $\Pr(S_j)$ are joint probability distributions for cluster C_i and sepset S_j respectively. Figure 3.6 shows an example of tree decompo-



(a) a 3-variable chain

(b) dynamic programming

(c) heuristic search

Figure 3.5: An illustration of two M-Modes algorithms solving a toy 3-variable chain problem ($\delta = 1$). (a) is the chain graph with 3 variables and each variable has 2 labels. (b) A diagram showing how dynamic programming algorithm works. The rounded corners rectangle are δ-subgraphs, inside them are their local MAPs, where red labels are boundary and orange are interior variables. Two consistent combinations form 2 modes. (c) The heuristic search tree. Each node is the search state (labeling), where red labels are different boundary variables forming branches, and orange are calculated or current fixed labels. It kills the state if it cannot pass a local MAP test. The surviving results at leaf nodes are modes.

sition. We can do MAP inference using Max-Product belief propagation (Wainwright and Jordan, 2008) to seek the minimum energy. The clusters should be ordered before belief propagation (e.g. performing a depth-first traversal; see Figure 3.6(b) red labels). Sepsets are omitted, as sepsets are always subsets of its adjacent clusters in a junction tree.

3.3.1 δ-subtrees

The δ-subgraphs are the basic search units of the existing M-Modes search algorithm. To utilize tree decomposition in the search, we need to map each δ-subgraph onto the decomposition. Since a δ-subgraph is always a connected graph, the corresponding part of the junction tree that the subgraph spans must be contiguous; we call the cluster(s) which the δ-subgraph spans as a *δ-subtree*. In order not to waste computation, we should find

Table 3.2: Evolution of algorithms solving M-Modes on chain, tree, and general loopy graphs. DP: Dynamic Programming; HS: Heuristic Search; TD: Tree Decomposition.

Chain	**Tree**	**Loopy**
DP (Chen et al., 2013)	DP (Chen et al., 2014)	–
–	HS (Chen, Yuan, and Chen, 2016)	HS + TD (Chen et al., 2018)

a minimum sub junction tree that adequately represents a δ-subgraph for a local MAP calculations.

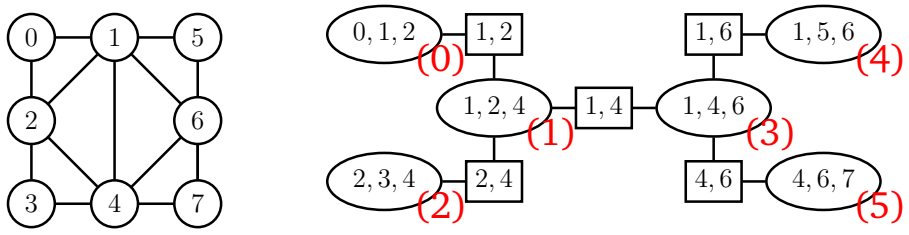
Definition 1 (δ-subtree)**.** *For a δ-subgraph S_δ, the δ-subtree T_S is a minimum sub junction tree that includes all the vertices in $cl(S)$.*

There is only one such δ-subtree for each δ-subgraph. Theorem 2 claims that collecting the clusters which contain the interiors is sufficient to obtain a δ-subtree, which would automatically include the boundaries. The proof is omitted, which can be found at Chen et al. (2018) Theorem 5. Table 3.3(a) shows an example of constructing corresponding δ-subtrees from each δ-subgraphs. At each search step, it is necessary to perform a local MAP inference at the corresponding δ-subtrees.

Theorem 2 (Finding δ-subtree)**.** *A δ-subtree satisfies (1) its clusters contain all the interior variables, and (2) each cluster contains at least one interior variable.*

3.3.2 Coordinating Cluster and δ-subtree Ordering

Instead of δ-subgraphs with vertex, the tree decomposition implementation takes a list of δ-subtrees as input. The heuristic search layers correspond to different δ-subtrees. The heuristic function is calculated in the reverse order of the cluster ordering. So δ-subtree ordering should be coordinated with cluster ordering. This coordination can be solved



(a) an 8-variable graph

(b) Tree decomposition

Figure 3.6: An example of Tree Decomposition: (a) the original graphical model and (b) the junction tree. The ellipses are clusters, and the rectangles are sepsets. The red labels at each clusters are the cluster ordering.

via sorting by the lexical ordering, with respect to the global cluster ordering, of each δ-subtree. See Table 3.3(b).

Solving M-Modes by tree decomposition allows a sequence of local subgraphs to be mapped to a set of subtrees sweeping through the tree decomposition, thus enabling a smooth and efficient transition back and forth between mode search, heuristic function calculation, and local MAP calculations.

3.4 Optimizing Heuristic Search Ordering

The most recent updates of this work are from Chen, Yuan, and Chen (2020), which present methods for finding optimized heuristic search orderings for solving M-Modes. We use a simple example to explain the impact of search orderings on the size of search spaces. We define a metric for evaluating the quality of search orderings and present related algorithms for finding optimized orderings.

Example 1. *Different search orderings for a tree graph in Figure 3.7:*

Table 3.3: An example of δ-subgraphs and δ-subtrees on the graph from Figure 3.6 ($\delta = 1$). (a) **Index** are the indices assigned to all δ-subgraphs, $\mathcal{S}_\delta$ are the interior variables of each δ-subgraphs, $\partial\mathcal{S}_\delta$ are the boundary variables; (b) Coordinating δ-subtree ordering ($\vec{ord}$) with cluster ordering (red labels at Figure 3.6(b)). $\mathcal{T}_g$ are explored clusters, and $\mathcal{T}_h$ are unexplored clusters which should be calculated backward MAP as heuristic functions.

Index	$\mathcal{S}_\delta$	$\partial\mathcal{S}_\delta$	$\mathcal{T}_\mathcal{S}$
[0]	0	1, 2	(0)
[1]	1	0, 2, 4, 5, 6	(0)(1)(3)(4)
[2]	2	0, 1, 3, 4	(0)(1)(2)
[3]	3	2, 4	(2)
[4]	4	1, 2, 3, 6, 7	(1)(2)(3)(5)
[5]	5	1, 6	(4)
[6]	6	1, 4, 5, 7	(3)(4)(5)
[7]	7	4, 6	(5)

(a)

$\vec{ord}$	$\mathcal{T}_\mathcal{S}$	$\mathcal{T}_g$	$\mathcal{T}_h$
[0]	(0)	(0)	(1)(2)(3)(4)(5)
[2]	(0)(1)(2)	(0)(1)(2)	(3)(4)(5)
[1]	(0)(1)(3)(4)	(0)(1)(2)(3)(4)	(5)
[4]	(1)(2)(3)(5)	(0)(1)(2)(3)(4)(5)	–
[3]	(2)	–	–
[6]	(3)(4)(5)	–	–
[5]	(4)	–	–
[7]	(5)	–	–

(b)

A Naive Search Ordering

We now use a simple example to illustrate how search orderings impact the size of search spaces. We use the tree graph in Figure 3.7 as our example for easier illustration; the same idea easily generalizes to loopy graphs by junction tree representation. Assume that all the vertices have label size 2, and $\delta = 1$. First, we use a depth-first traversal to get an ordering of vertices which is $0 \to 1 \to 2 \ldots \to 8 \to 9$. We then create an ordering of all the δ-subgraphs such that the interior variables of the subgraphs follow the same ordering (See Column $\mathcal{S}_\delta$ in Table 3.4(a)). Then, for each δ-subgraphs, we circumscribe it and get boundary variables (See Column $\partial\mathcal{S}_\delta$ in Table 3.4(a)). For example, the boundary of δ-subgraph $\{2\}$ are $\{1, 3, 7, 8\}$. Fixing the labels of the boundary, there is a unique local

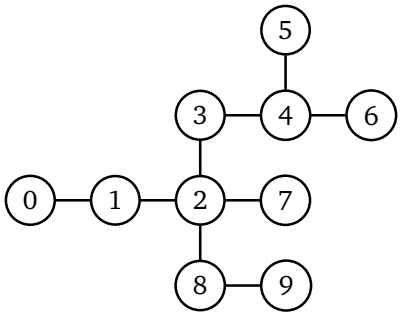


Figure 3.7: A tree graph

MAP for the interior variable $\{2\}$.

We first perform M-Modes search using the δ-subgraph ordering $\mathcal{S}_{\delta}^{+}[o\vec{r}d_1]$ in Table 3.4(b). Initially, none of the vertices are instantiated. For the first layer, the δ-subgraph has interior 0 and the new boundary 1. We create two successor search nodes corresponding to the two labels of vertex 1 respectively. We also compute the local MAP value for the interior 0 conditional on the value of boundary 1. Afterward, we may have two search nodes with partial labelings such as $\langle 10 \rangle$ and $\langle 01 \rangle$. Each of the nodes has a score of summing the energies of fixed vertices (as current cost) and undecided vertices (as heuristic). We choose the search node with the lowest value to continue.

The δ-subgraph corresponding to the next layer has interior 1 and boundaries $\{0, 2\}$. But vertices 0 and 1 are already instantiated in the previous step; the only vertex 2 is a new boundary. We, therefore, instantiate 2 to different values and *verify* whether the value of interior 1 is the local MAP or not. We only create a successor node if it passes the verification. Out of the two possible successor nodes with partial labelings $\langle 100 \rangle$ and $\langle 101 \rangle$, we may find that only the second one passed the verification and is kept for further search.

Then, the next layer has 3 new boundaries, so we need to consider all their combinatorial configurations, resulting in up to $2^3 = 8$ new successor nodes. The search continues in

Table 3.4: δ-subgraphs for the tree graph in Figure 3.7 where $\delta = 1$; (a) **Index** are the indices assigned to all δ-subgraphs, $\mathcal{S}_\delta$ are the interior variables of each δ-subgraphs, $\partial\mathcal{S}_\delta$ are the boundary variables; (b) $\mathcal{S}_\delta^+[\vec{ord}_1]$ are the frontiers given the default (naive) ordering $\vec{ord}_1$; (c) $\mathcal{S}_\delta^+[\vec{ord}_2]$ are the frontiers given an optimized ordering $\vec{ord}_2$.

Index	$\mathcal{S}_\delta$	$\partial\mathcal{S}_\delta$
[0]	0	1
[1]	1	0, 2
[2]	2	1, 3, 7, 8
[3]	3	2, 4
[4]	4	3, 5, 6
[5]	5	4
[6]	6	4
[7]	7	2
[8]	8	2, 9
[9]	9	8

(a)

$\vec{ord}_1$	$\mathcal{S}_\delta^+[\vec{ord}_1]$
[0]	1
[1]	2
[2]	3, 7, 8
[3]	4
[4]	5, 6
[5]	–
[6]	–
[7]	–
[8]	–
[9]	9

(b)

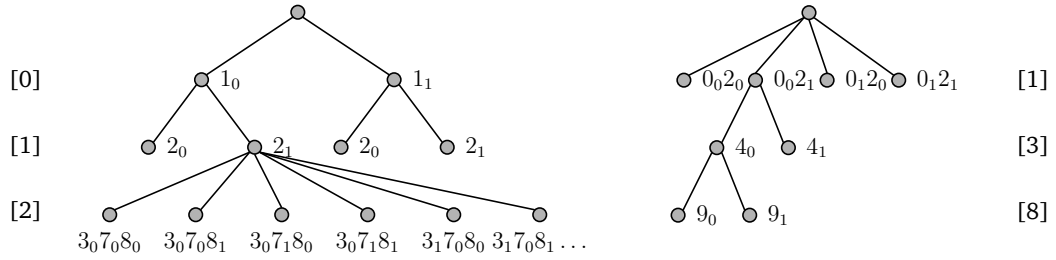
$\vec{ord}_2$	$\mathcal{S}_\delta^+[\vec{ord}_2]$
[1]	0, 2
[3]	4
[8]	9
[0]	–
[5]	–
[6]	–
[4]	–
[7]	–
[2]	–
[9]	–

(c)

the same way for the remaining δ-subgraphs. If all of the δ-subgraphs have been checked and a search path survived, the final labeling is a mode. If we use A* algorithm, the first mode found must be the 1-Mode, which is also the MAP. And the second mode must be the 2-Mode, etc. We stop when we get enough modes. See Figure 3.8(a) for the partial search tree created above.

A Better Search Ordering

Now consider another δ-subgraph ordering in Table 3.4(c). We start by searching the δ-subgraph with interior 1 and boundaries $\{0, 2\}$. Up to four search nodes corresponding



(a) Default Ordering

(b) Optimized Ordering

Figure 3.8: Two example search trees (first three layers) for solving M-Modes for the tree graph in Figure 3.7 when $\delta = 1$. The layer indices are the search ordering (same as the interiors) and the labels for each nodes are frontier labelings (the big numbers are the vertices and the subscripts are chosen labels). (a) is the search tree for default ordering and (b) is for the optimized ordering.

to partial labelings $\langle 0\_0 \rangle$, $\langle 0\_1 \rangle$, $\langle 1\_0 \rangle$ and $\langle 1\_1 \rangle$ will be created. The value of interior 1 is filled in by local MAP inference. Then for the next layer, since vertex 2 is already fixed, we skip searching the δ-subgraph with interior 3 and new boundary 4, in which we branch on the values of 4 and get values for 3 by local MAP inference. Then, we skip searching the δ-subgraph with interior 8 and boundary 9. After only three search layers, we already obtained values for all vertices except $5, 6, 7$, whose values can be obtained by local MAP inference as well. Other remaining δ-subgraphs whose interiors did not obtain their values via local MAP inference needs to be verified whether the values are local MAPs or not. Therefore, all these remaining searches form one single search path downward and do not increase the number of branches. We call these no-branching layers as *verification* layers for simplicity. Comparing to the naive search ordering, the new ordering leads to much smaller search space. See Figure 3.8(b) for part of the new search tree.

3.4.1 Frontiers

We now formalize some observations from the simple example. At any step, the variables involved in the search are either already instantiated in previous search steps, or new and yet to be instantiated. Such new variables are either interior or boundary variables. When the boundary variables of a δ-subgraph are instantiated to one particular configuration, the interiors have to take on the values of the local MAP to be eligible for further search. Therefore, only the undecided *new* boundaries contribute to increasing the size of search space, and interiors do not. The number of new successor nodes is exponential in the number of new boundary variables. We call these undecided boundaries *frontiers*.

Definition 2 (Frontier)**.** *Vertex v is a frontier, denoted as $v \in S_\delta^+[i]$ if and only if $v \in \partial S_\delta[i]$, and $v \notin cl(S_\delta[j])$ for all $j < i$.*

Frontiers are highly specific to particular δ-subgraph orderings. If a variable was first searched as an interior variable, it has no chance to become frontiers. In Tables 3.4(b) and 3.4(c), $\mathcal{S}_\delta$ denotes the set of all the δ-subgraphs. This set can be ordered by an ordering vector $\vec{ord}$, denotes as $\mathcal{S}_\delta[\vec{ord}]$. The ordering ord_1 in Table 3.4(b) introduced 8 frontiers in total, while ord_2 in Table 3.4(c) introduced only 4 frontiers in total.

3.4.2 Defining Search Space Complexity

The simple example above shows that different search orderings lead to search spaces with drastically different sizes. We define a metric to measure the search space complexity induced by a search ordering. Given a particular ordering of δ-subgraphs $\mathcal{S}_\delta[\vec{ord}]$, we denote the induced search tree as $\mathrm{SearchTree}_\delta[\vec{ord}]$. Traditionally, we use the *size of the search tree* as the complexity of the search space, which is exponential in the depth of the tree. However, because the number of search layers of M-Modes is always equal to the number of δ-subgraphs, but many of them form verification layers without branching, we

propose to use *the number of leaves* to measure the size of the search space. The tree size can be bounded by the leaf size as follows:

$$\left|\text{SearchTree}_\delta[\vec{ord}]\right| = c \cdot \left|\text{Leaf}(\mathcal{S}_\delta[\vec{ord}])\right|, \ (\ 2 < c < |S_\delta| - 1\ ) \tag{3.1}$$

Here the coefficient c is greater than 2 because there must exist at least one verification layer. And it is less than $|S_\delta| - 1$ because there must exist at least one layer with frontiers.

Therefore, based on the fact that only frontiers increase the number of search branches, the number of leaves is equal to the total number of branches induced by the combinatorial configurations of the frontiers. We get:

$$\left|\text{Leaf}(\mathcal{S}_\delta[\vec{ord}])\right| = \prod_{i=0}^{\left|\mathcal{S}_\delta^+[\vec{ord}]\right|-1} \prod_{v \in S_\delta^+[i]} L_v = \prod_{v \in \mathcal{S}_\delta^+[\vec{ord}]} L_v \tag{3.2}$$

3.4.3 Finding Optimized Search Ordering

Once the search space complexity is defined, we find an optimized ordering by searching for the ordering which minimizes the leaf size. Taking log transforms the multiplicative total size into a summation of log label sizes.

Problem 1 (Finding Optimized Ordering)**.** *Find* $\vec{ord}$ *s.t.*

$$\arg\min_{\vec{ord}} \sum_{v \in \mathcal{S}_\delta^+[\vec{ord}]} \log(L_v) \tag{3.3}$$

The new objective function is decomposable. We, therefore, formulate finding the optimal search ordering as a *shortest-path* problem. We start from an empty ordering and adds one δ-subgraph at each step until all δ-subgraphs have been added to the ordering; an accumulative total size is maintained throughout. Given the ordering, search is only

a subroutine of mode search, and there are $|\mathcal{S}_\delta|!$ possible solutions, we cannot afford to solve this problem optimally. We, therefore, chose the depth-first search as we can stop it anytime to output the best ordering found thus far.

The depth-first search works as follows. At each step, we add the δ-subgraph which not only *overlaps* with an existing δ-subgraph, if exists, but also increases the total size *maximally*, with ties broken arbitrarily. This design of the search step makes sure that no gap is introduced between δ-graphs in the (partial) ordering throughout the search. Once tbe current δ-subgraphs cover all vertices, the remaining δ-subgraphs can be appended in any order as verification layers because they add zero cost to the total size, producing a complete search ordering. We can run the depth-first search for as long as allowed and output the best subgraph ordering found in the end.

Finding a search ordering is only a preprocessing step for the actual M-Modes search. We cannot afford to spend too much time here. If the number of δ-subgraphs is large, solving the shortest-path problem may take a long time, because there are $|\mathcal{S}_\delta|!$ possible solutions. In the following, we present two pruning criteria to speed up the ordering search. First, because only δ-subgraphs that have frontiers affect the size of the search space, we only need to consider adding δ-subgraphs that have undecided variables. All other δ-subgraphs are skipped. Once we have all vertices added as either frontier or interiors, we simply append remaining subgraphs in *any* order as verification layers, producing a complete search ordering. Note that we cannot skip any of these verification layers ascribed to the Global-Local Theorem. Second, because we want to find an optimized search ordering that is friendly for backward MAP inference, the δ-subgraphs with new boundary variables in the optimized ordering is kept consistent with their order of appearance in the naive ordering, i.e., consistent with the cluster ordering within the tree decomposition context. In other words, the former is a subsequence of the latter.

Example 2. *Different search orderings for tree decomposition in Figure 3.6 (See Table 3.5):*

Table 3.5: δ-subgraphs for the tree decomposition in Figure 3.6 where $\delta = 1$; (a) **Index** are the indices assigned to all δ-subgraphs, $\mathcal{S}_\delta$ are the interior variables of each δ-subgraphs, $\partial\mathcal{S}_\delta$ are the boundary variables; (b) $\mathcal{S}_\delta^+[o\vec{r}d_1]$ are the frontiers given the default (naive) ordering $o\vec{r}d_1$; (c) $\mathcal{S}_\delta^+[o\vec{r}d_2]$ are the frontiers given an optimized ordering $o\vec{r}d_2$.

Index	$\mathcal{S}_\delta$	$\partial\mathcal{S}_\delta$
[0]	0	1, 2
[1]	1	0, 2, 4, 5, 6
[2]	2	0, 1, 3, 4
[3]	3	2, 4
[4]	4	1, 2, 3, 6, 7
[5]	5	1, 6
[6]	6	1, 4, 5, 7
[7]	7	4, 6

(a)

$o\vec{r}d_1$	$\mathcal{S}_\delta^+[o\vec{r}d_1]$
[0]	1, 2
[2]	3, 4
[1]	5, 6
[4]	7
[3]	–
[6]	–
[5]	–
[7]	–

(b)

$o\vec{r}d_2$	$\mathcal{S}_\delta^+[o\vec{r}d_2]$
[0]	1, 2
[3]	4
[5]	6
[1]	–
[2]	–
[4]	–
[6]	–
[7]	–

(c)

3.4.4 δ-Vertex Cover Problem

Ignoring label size for simplicity, Optimizing search orderings for M-Modes is equivalent to solving a problem we call δ-vertex cover, formally define as follows.

Problem 2 (δ-Vertex Cover)**.** *A δ-vertex cover $\mathcal{V}'$ of an undirected graph $\mathcal{G}$ is a minimum subset of $\mathcal{V}$ such that for each $(\delta+1)$-subgraph, there is at least one vertex $v \in (\delta+1)$-subgraph and $v \in \mathcal{V}'$.*

The connection between δ-vertex cover and optimizing search ordering is straightforward: since there must be a vertex in the cover set for any $(\delta+1)$-subgraph, the cover set $\mathcal{V}'$ must decompose the graph into connected subgraphs with size no larger than δ. We can obtain an actual search ordering from $\mathcal{V}'$ as follows: We start with a connected subgraph

with size δ and create a corresponding δ-subgraph; the neighboring vertices belonging to $\mathcal{V}'$ automatically become its boundary variables (frontiers). Then for any connected subgraph adjacent to an existing δ-subgraph, if the subgraph has size δ, we directly map it to a δ-subgraph; otherwise, if its size is smaller than δ, we borrow as many vertices from already visited ones to create a full-size δ-subgraph. Any neighboring vertex belonging to $\mathcal{V}'$ become new frontiers. We repeat this step until all vertices are visited. The above procedure makes sure all the vertices in $\mathcal{V}'$ become frontiers in the search ordering.

The decision problem of δ-vertex cover is defined as: Does graph G have a δ-vertex cover of size at most k? It is clear that the standard vertex cover, an NP-Complete problem, is a special case with δ equal to 1. By reducing the vertex cover problem to δ-vertex cover, we can see the latter is also NP-complete.

Theorem 3 (δ-Vertex Cover Complexity)**.** *The δ-vertex cover problem is NP-complete.*

3.4.5 Remarks

Based on the observation that different search orderings for solving M-Modes have a huge impact on the size of search spaces, Chen, Yuan, and Chen (2020) propose methods for measuring the quality of search orderings and related algorithms for finding optimized search orderings. The proposed methods were shown to result in up to orders of magnitude speedups in the experiments.

An interesting observation from this research is that, in contrast to the common belief that search orderings only affect the *practical* performance, in M-modes problem, proper orderings also reduce the size of the search space. This raises new challenges and opportunities for the design of efficient heuristic search algorithms. We believe the investigation of M-modes problem would be a novel addition to the rich literature of heuristic search.

Algorithm 2 DP for Solving M-Modes

Input: G — A graphical model; δ — δ size; m — number of modes

Output: M – M-Modes solutions

function DP(G, δ, m)
  $\mathcal{S} \leftarrow$ COMPILE-SUBGRAPH-SET(G, δ), $\hat{\mathcal{L}} \leftarrow \varnothing$, $M \leftarrow \varnothing$
  for all $S \in \mathcal{S}$ **do** ▷ For each subgraph
    $\hat{L} \leftarrow \varnothing$
    for all $l \in D^{\partial S}$ **do** ▷ For each combinatorial boundary values
      $\hat{l} \leftarrow$ LOCAL-MAP(l)
      $\hat{L}.\mathrm{add}(\hat{l})$
    end for
    $\hat{\mathcal{L}}.\mathrm{add}(\hat{L})$
  end for
  for all combinatorial $\hat{l}_{1,2,\ldots,|\mathcal{S}|} \in \hat{\mathcal{L}}$ **do**
    $x \leftarrow \{\hat{l}_1, \hat{l}_2, \ldots\}$
    if CONSIST(x) **then** $M.\mathrm{add}(x)$ ▷ Connect consistent local MAPs from each subgraphs
    end if
  end for
  return $M.\mathrm{top}(m)$
end function

Algorithm 3 M-A* for Solving M-Modes

Input: G — a graphical model; δ — δ size; m — number of modes

Output: M – M-Modes solutions

function M-A*(G, δ, m)
  $\mathcal{S} \leftarrow$ COMPILE-SUBGRAPH-SET(G, δ), $Q \leftarrow \varnothing$, $M \leftarrow \varnothing$ ▷ Q is the open list
  Q.push($\{ i \leftarrow 0, g \leftarrow 0, h \leftarrow$ MAP(G), $l \leftarrow \varnothing \}$)
  repeat
    $cur \leftarrow Q$.pop()
    if $cur.i = |\mathcal{S}|$ **then**
      M.add($cur.l$)
      if $|M| = m$ **then return** M
      end if
    else
      $succs \leftarrow \varnothing$, $S \leftarrow \mathcal{S}_{i+1}$
      for all $l \in D^{\partial S}$ **do** ▷ For each combinatorial boundary values of next subgraph
        $\hat{l} \leftarrow$ LOCAL-MAP(l)
        if CONSIST($cur.l$, $\hat{l}$) **then**
          $l' \leftarrow cur.l + \hat{l}$
          $succs$.add($\{ i \leftarrow i + 1, g \leftarrow g(l'), h \leftarrow h(l'), l : l' \}$)
        end if
      end for
      Q.push($succs$)
    end if
  until $Q = \varnothing$
end function

Chapter 4

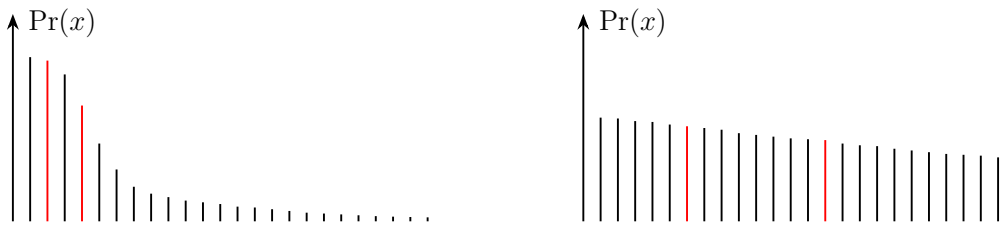
Min-Loss Multiple Inference

4.1 Minimizing Expected Loss

4.1.1 Motivation

There is no solid proof that finding these posterior modes can always luckily obtain good results by applying some diversity constraints over with high probability points. Both Diverse M-Best and M-Modes have critical drawbacks in that they require tuning diversity parameters, namely either λ or δ, by cross-validation. As λ is a continuous numeric, we cannot easily find an optimal value for all problems in Diverse M-Best. The δ in M-Modes is an integer value, but its optimum value is varying for different cases. If λ or δ is set too low, the next solution may still be trapped at the same peak; if too high, many good solutions will be ignored. This problem is related to the topology of the distribution landscape. Consequently, these methods do not consider the landscape as a whole picture.

This raises a concern about choosing solutions on different distribution landscapes. See Figure 4.1. When the landscape is steep, where some of the solutions have very



(a) Steep (b) Flat

Figure 4.1: Suggested multiple solutions (red bars) for steep and flat probability distribution landscapes.

high probabilities, it is more profitable to select from the peak and sacrifice diversity (Figure 4.1(a)); when the landscape is flat, where all of the labelings have similar probabilities, more diverse choices would give higher chance to be close to the ground truth (Figure 4.1(b)). In addition, for M-Modes solutions, if there is only one big hill, e.g., many models assuming Gaussian distribution, the MAP is its only mode solution. We cannot generate more candidates.

Therefore, we should tune the hyper-parameter automatically. We agree with the opinion that diversity should not be explicitly enforced ad hoc, and should be an emerging property (Dey et al., 2015) reflecting different distribution landscapes. Bayesian methods are characterized by the use of distributions to summarize data and draw inferences. Directly and jointly optimizing oracle accuracy for M solutions by more general Bayesian methods should be much better than diversifying the modes. Chen et al. (2020) create a more fundamental parameter-free approach.

4.1.2 Expected Loss

Given any target labeling y for prediction, the *expected loss* or *loss* of y is the accumulation of the distance of each labeling y_{gt} from y weighted by probability of y_{gt}, i.e., the dot

product of distances and probabilities:

$$\mathsf{loss}(y) = \mathop{\mathbb{E}}_{y_{\mathrm{gt}} \sim \mathcal{Y}} \left[\, \Delta(y, y_{\mathrm{gt}}) \, \right] \tag{4.1}$$

$$= \sum_{y' \in \mathcal{Y}} \Pr(y') \cdot \Delta(y, y') \tag{4.2}$$

So that, if a generated data set has an enough large number of samples, the labelings' expected error rate (distance) should converge to this expected loss (See Table 1 col. S∞ at Chen et al. (2020)).

Based on the concept of expected loss, we define another useful optimization inference task called *Min-Loss MAP*:

Problem 3 (Min-Loss MAP)**.** *Find a labeling of a model which has lowest expected loss:*

$$y^* = \arg\min_{y \in \mathcal{Y}} \mathsf{loss}(y) \tag{4.3}$$

The solution of Min-Loss MAP can be quite different from MAP. MAP can be considered as a special case of Min-Loss MAP in which $\Delta(y, y_{\mathrm{gt}}) = [\![\, y \neq y_{\mathrm{gt}} \,]\!]$.

Common distance measures are nodewise distance, thus we can utilize this property to efficiently minimize each variable's distance. Then summing over these choices is naturally the global minimum.

Hamming distance is a nodewise distance, and it counts distances of different variables independently. Hence finding a minimal distance label of a variable (i.e. summation of other labels has minimum probability) is the same as a maximal marginal (i.e. current label has maximum probability). We can accordingly conclude such a theorem that max marginals can exactly solve the Min-Loss MAP problem using Hamming distance.

Theorem 4. *Max marginals $\Leftrightarrow$ Hamming Min-Loss MAP.*

Despite max marginals have high accuracy is a known observation in the literature, we should notice that this split rule can only be applied for Hamming loss.

(a) Average Accuracy

(b) Oracle Accuracy

Figure 4.2: Diagrams shows the different measurements between (a) Average Accuracy and (b) Oracle Accuracy for $M = 2$. The red nodes represent the predictions and the black circles represent other labelings. The lines between red and black nodes represent the ways for distance measurement. For average accuracy, the loss takes all the labelings into account, while for oracle accuracy, it only measures the distances of closer predictions.

4.2 Min-Loss M-Best

Similar to M-Best multiple inference problems, we can also extend our Min-Loss MAP task to *Min-Loss M-Best*, which computes the top M solutions with the lowest expected loss. But it is not the correct way for oracle accuracy when $M > 1$. Oracle accuracy considers the highest prediction as to the distance measurement; while naive Min-Loss M-Best considers each prediction to the whole distribution, i.e. average accuracy. See Figure 4.2 for a better understanding of the difference. Therefore, optimizing oracle accuracy for Min-Loss M-Best problem is totally another story.

4.2.1 Min-Loss M-Best For Oracle Accuracy

Formally, optimizing oracle accuracy for M-solution problem by minimizing expected loss can be described as:

$$\{y\}_M^* = \arg\max_{\{y\}_M \in \mathcal{Y}} \mathsf{OrcAcc}\big(\{y\}_M\big) = \arg\min_{\{y\}_M \in \mathcal{Y}} \mathsf{loss}(\{y\}_M) \tag{4.4}$$

$$= \arg\min_{\{y\}_M \in \mathcal{Y}} \mathop{\mathbb{E}}_{y_{\text{gt}} \sim \mathcal{Y}} \Big[\min\big(\Delta(y_1, y_{\text{gt}}), \ldots, \Delta(y_M, y_{\text{gt}})\big) \Big] \tag{4.5}$$

$$= \arg\min_{\{y\}_M \in \mathcal{Y}} \sum_{y' \in \mathcal{Y}} \Pr(y') \Big[\min\big(\Delta(y_1, y'), \ldots, \Delta(y_M, y')\big) \Big] \tag{4.6}$$

This problem aims to jointly find M solutions, $\{y\}_M$, which at least one of them has the lowest expected loss.

As the whole labeling distribution $\mathcal{Y}$ is exponentially large, we cannot find an exact algorithm to solve the Min-Loss MAP problem in a large model. However, it is possible to solve it approximately using M-Best solutions, i.e., let $\{y\}_{m_{\text{(Best)}}} \sim \mathcal{Y}$.

Once we have generated $m_{\text{(Best)}}$ solutions, we can form a pairwise loss table where each entry is a term that is the row's probability multiplies the distances of two solutions. For conciseness, we simplified the terms: $\Pr(y_m)$ as p_m, and $\Delta(y_m, y_n)$ as $\Delta_{m,n}$. This is a pre-processing step. See Figure 4.3. Note that the diagonal entries are zero as $\Delta_{m,m} = 0$.

We can formulate finding an oracle accuracy M solutions as: Minimize the summation of all the entries such that 1) only one entry for each row can be selected and 2) the number of correlated columns should at most M. For no ambiguity, we use m for the input table size (from M-Best), and M for the number of solutions of oracle accuracy. We use *at most* since that it is acceptable to use fewer solutions to get the same results.

	y_1	y_2	$\cdots$	y_{m-1}	$y_{m_{\text{(Best)}}}$
y_1	0	$\underline{p_1\Delta_{12}}$	$\cdots$	$p_1\Delta_{1,m-1}$	$p_1\Delta_{1,m}$
y_2	$p_2\Delta_{21}$	$\underline{0}$	$\cdots$	$p_2\Delta_{2,m-1}$	$p_2\Delta_{2,m}$
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$
y_{m-1}	$p_{m-1}\Delta_{m-1,1}$	$p_{m-1}\Delta_{m-1,2}$	$\cdots$	$\underline{0}$	$p_{m-1}\Delta_{m-1,m}$
$y_{m_{\text{(Best)}}}$	$p_m\Delta_{m,1}$	$\underline{p_m\Delta_{m,2}}$	$\cdots$	$p_m\Delta_{m,m-1}$	0

Figure 4.3: Pairwise Loss Table from M-Best. The red shaded two columns are two chosen labelings (y_2 and y_{m-1}). The underlined terms are weighed distances to each labelings (rows). The target optimizating problem is to minimize the summation of these terms.

After compiling the table, we can formally define the optimization problem as follows:

$$\min \sum_{i=1}^{m}\sum_{j=1}^{m} \mu_{ij} \cdot (p_i\Delta_{ij}) \tag{4.7}$$

$$\text{s.t.} \qquad \mu_{ij} \in \{0,1\}, \quad \forall i,j \tag{4.8}$$

$$\sum_{j=1}^{m} \mu_{ij} = 1, \quad \forall i \tag{4.9}$$

$$\sum_{j=1}^{m}\prod_{i=1}^{m}(1-\mu_{ij}) \geqslant m - M, \quad \forall i,j \tag{4.10}$$

Our target is to optimize over the indicator variables μ, with the constraints that force each row to choose only one entry.

The product of the last constraint simulates an OR constraint, indicating the number of involving columns (labelings chosen) should be fewer than M. See Figure 4.3's red shades. Because the last constraint is not linear, we cannot make it an integer linear programming problem.

4.2.2 Searching Optimal Min-Loss M-Best

We propose a search method: For small M, we can exhaustively search the possible choices y_M from top m solutions. And for each choice, we compute its oracle accuracy with respect to the m solutions. We report the choices with minimum expected loss, y^*_M. When we accumulate the distance, we prefer to expand labelings that have high probabilities. At the same time, the current best solution is used as an upper bound to prune rest computation which has a larger value than the bound. See Algorithm 4. This method has complexity $\mathcal{O}(\binom{m}{M}vmM)$.

When M is large, we would like to randomly sample M solutions out of m instead of exhaustively enumerating all the possible choices. In addition, we also use the current best result to do the pruning, then run the method for a certain amount of time and report the current best choices we found.

One interesting question is whether some heuristic functions can help us speed up the search. Since getting more labelings will obtain lower distances, searching optimal Min-loss M-Best is not easy to be formulated as shortest path problems. One tentative way is to formulate the problem from m choosing M to m not choosing $(m - M)$, but it also raised another problem that the search space is even larger (because $m \gg M$). Finding better heuristic search algorithms to solve this problem is interesting but left as future work.

4.3 Remarks

This work developed and evaluated a fundamental formulation for multiple inference, as a Bayesian method approach, by directly optimizing oracle accuracy via expected loss. The top advantage of the new method is that it is parameter-free, in contrast to other MAP estimation diverse approaches. We demonstrated that this idea is clearly effective.

Although promising, the proposed methods are currently restricted by approximately using top M-Best solutions to simulate the whole distribution. If the model is very large, top M-Best solutions may not be able to depict the whole distribution, as the "long tail" phenomena happen. But expanding beyond top M solutions will lead this problem to be much harder to solve. More advanced techniques are yet to be developed. For example, when using nodewise distance measures, it is worth investigating whether we can continue utilizing this splittable property (as Theorem 4) either to prune some values or divide and conquer the problem in a much more intelligent way.

Algorithm 4 Finding Min-Loss M-Best Solutions

Input: M-Best solutions: $\{y\}_m$; M

Output: Optimal M predictions: $\{y\}^*_M$

function MIN-LOSS M-BEST($\{y\}_m$, M)

$\{y\}_m$.sort(on = Pr)

$loss_{\min} \leftarrow \infty$

$\mathcal{Y}_M \leftarrow$ Combination($\{y\}_m, M$)

or Sample($\{y\}_m, M$)

for all $\{y\}_M \in \mathcal{Y}_M$ **do**

$loss \leftarrow 0$

for all $y_m \in \{y\}_m$ **do**

$loss \leftarrow loss + p_m \cdot \min(\Delta_{m,M^1}, \ldots, \Delta_{m,M})$

if $loss \geqslant loss_{\min}$ **then break**

end if

end for

if $loss < loss_{\min}$ **then**

$loss_{\min} = loss, \{y\}^*_M = \{y\}_M$

end if

end for

return $\{y\}^*_M$

end function

Chapter 5

Conclusions

In this work, we study different approaches for solving multiple inference. Here is the road map to illustrate the terms and the evolution of these methods. See Figure 5.1. We start from going over the background of MAP inference, to introduce multiple inference. Multiple inference evolves from M-Best, Diverse M-Best, M-Modes, till Min-Loss M-Best. We focus much on developing current M-Modes algorithms and proposing a new Min-Loss M-Best idea. We detail solutions of solving M-Modes from Dynamic Programming to Heuristic Search with optimized orderings and tree decomposition for general graphs.

Existing approaches are still restricted by the complexity of combinatorial optimizations. Future works are to develop new techniques to efficiently solve these problems. In addition, Successful ideas can inspire works in other fields such as learning diverse networks (Chen and Yuan, 2019).

We also need more applications to verify the correctness of proposed multiple inference ideas in graphical models. Real-world applications may be involved networks with hundreds of nodes. A certain extent of approximation could also be tolerant for speeding up calculating large-scale networks.

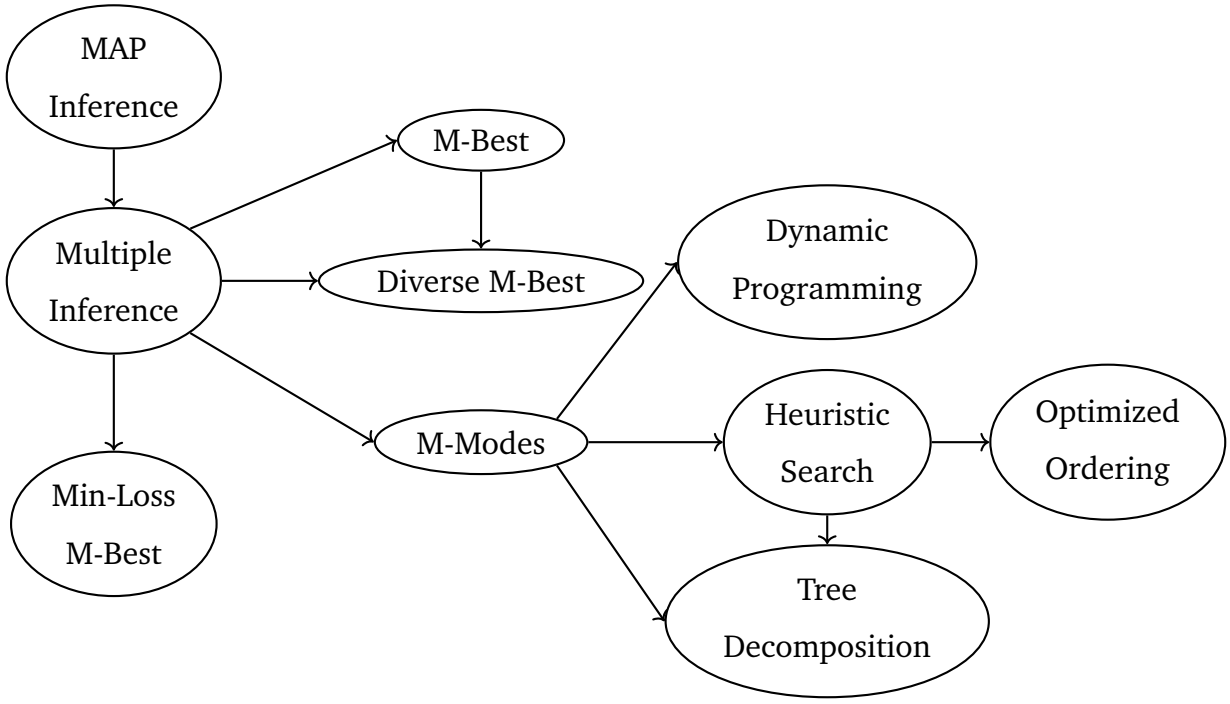


Figure 5.1: Road Map of Solving Multiple Inference in Graphical Models

Bibliography

Andres, B.; Beier, T.; and Kappes, J. H. 2012. Opengm: A c++ library for discrete graphical models. *arXiv preprint arXiv:1206.0111*.

Batra, D.; Yadollahpour, P.; Guzman-Rivera, A.; and Shakhnarovich, G. 2012. Diverse M-best solutions in markov random fields. *Computer Vision–ECCV 2012* 1–16.

Bousquet, O., and Bottou, L. 2008. The tradeoffs of large scale learning. In Platt, J. C.; Koller, D.; Singer, Y.; and Roweis, S. T., eds., *Advances in Neural Information Processing Systems 20*. Curran Associates, Inc. 161–168.

Boykov, Y.; Veksler, O.; and Zabih, R. 2001. Fast approximate energy minimization via graph cuts. *PAMI* 23(11):1222–1239.

Chen, Y., and Tian, J. 2014. Finding the k-best equivalence classes of bayesian network structures for model averaging. In *AAAI*, 2431–2438.

Chen, C., and Yuan, C. 2019. Learning diverse bayesian networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 7793–7800.

Chen, C.; Kolmogorov, V.; Zhu, Y.; Metaxas, D.; and Lampert, C. H. 2013. Computing the M most probable modes of a graphical model. In *International Conf. on Artificial Intelligence and Statistics (AISTATS)*.

Chen, C.; Liu, H.; Metaxas, D.; and Zhao, T. 2014. Mode estimation for high dimensional

discrete tree graphical models. In *Advances in neural information processing systems*, 1323–1331.

Chen, C.; Yuan, C.; Ye, Z.; and Chen, C. 2018. Solving m-modes in loopy graphs using tree decompositions. In *International Conference on Probabilistic Graphical Models*, 145–156.

Chen, C.; Yang, J.; Chen, C.; and Yuan, C. 2020. Solving multiple inference by minimizing expected loss. In *The 10th International Conference on Probabilistic Graphical Models (PGM)*.

Chen, C.; Yuan, C.; and Chen, C. 2016. Solving m-modes using heuristic search. In *IJCAI*, 3584–3590.

Chen, C.; Yuan, C.; and Chen, C. 2020. Efficient heuristic search for m-modes inference. In *The 10th International Conference on Probabilistic Graphical Models (PGM)*.

Dechter, R., and Mateescu, R. 2007. And/or search spaces for graphical models. *Artificial intelligence* 171(2-3):73–106.

Dechter, R., and Pearl, J. 1985. Generalized best-first search strategies and the optimality of a. *Journal of the ACM (JACM)* 32(3):505–536.

Dechter, R., and Rish, I. 2003. Mini-buckets: A general scheme for bounded inference. *Journal of the ACM (JACM)* 50(2):107–153.

Dechter, R.; Flerova, N.; and Marinescu, R. 2012. Search algorithms for m best solutions for graphical models. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, AAAI'12, 1895–1901. AAAI Press.

Dechter, R. 1999. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence* 113(1-2):41–85.

Dey, D.; Ramakrishna, V.; Hebert, M.; and Andrew Bagnell, J. 2015. Predicting multiple structured visual interpretations. In *Proceedings of the IEEE International Conference on Computer Vision*, 2947–2955.

Flerova, N.; Marinescu, R.; and Dechter, R. 2016. Searching for the m best solutions in graphical models. *Journal of Artificial Intelligence Research* 55:889–952.

Flerova, N.; Rollon, E.; and Dechter, R. 2012. Bucket and mini-bucket schemes for m best solutions over graphical models. In *Graph Structures for Knowledge Representation and Reasoning*. Springer. 91–118.

Fromer, M., and Yanover, C. 2009. Accurate prediction for atomic-level protein design and its application in diversifying the near-optimal sequence space. *Proteins: Structure, Function, and Bioinformatics* 75(3):682–705.

Gimpel, K.; Batra, D.; Dyer, C.; and Shakhnarovich, G. 2013. A systematic exploration of diversity in machine translation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 1100–1111.

Hoffman, W., and Pavley, R. 1959. A method for the solution of the nth best path problem. *J. ACM* 6(4):506–514.

Huang, C., and Darwiche, A. 1996. Inference in belief networks: A procedural guide. *International journal of approximate reasoning* 15(3):225–263.

Kirillov, A.; Savchynskyy, B.; Schlesinger, D.; Vetrov, D.; and Rother, C. 2015a. Inferring M-Best diverse labelings in a single one. In *IEEE International Conference on Computer Vision (ICCV)*. IEEE.

Kirillov, A.; Savchynskyy, B.; Schlesinger, D.; Vetrov, D.; and Rother, C. 2015b. Inferring m-best diverse labelings in a single one. In *Proceedings of the IEEE International Conference on Computer Vision*, 1814–1822.

Kolmogorov, V., and Zabin, R. 2004. What energy functions can be minimized via graph cuts? *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 26(2):147–159.

Ladicky, L.; Russell, C.; Kohli, P.; and Torr, P. 2010. Graph cut based inference with co-occurrence statistics. *ECCV*.

Lauritzen, S. L., and Spiegelhalter, D. J. 1988. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)* 157–224.

Lawler, E. L. 1972. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science* 18(7):401–405.

Li, F.; Carreira, J.; and Sminchisescu, C. 2010. Object recognition as ranking holistic figure-ground hypotheses. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, 1712–1719. IEEE.

Meltzer, T.; Yanover, C.; and Weiss, Y. 2005. Globally optimal solutions for energy minimization in stereo vision using reweighted belief propagation. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 1, 428–435. IEEE.

Murphy, K. P. 2012. *Machine learning: a probabilistic perspective*. MIT press.

Nilsson, D. 1998. An efficient algorithm for finding the m most probable configurations in probabilistic expert systems. *Statistics and Computing* 8(2):159–173.

Nowozin, S., and Lampert, C. 2010. Structured learning and prediction in computer vision. *Foundations and Trends in Computer Graphics and Vision* 6(3-4):185–365.

Pearl, J. 1982. *Reverend Bayes on inference engines: A distributed hierarchical approach*. Cognitive Systems Laboratory, School of Engineering and Applied Science, University of California, Los Angeles.

Pearl, J. 1988. Probabilistic reasoning in intelligent systems. palo alto. *Morgan Kaufmann. PEAT, J., VAN DEN BERG, R., & GREEN, W.(1994). Changing prevalence of asthma in australian children. British Medical Journal* 308:1591–1596.

Prasad, A.; Jegelka, S.; and Batra, D. 2014. Submodular meets structured: Finding diverse subsets in exponentially-large structured item sets. In *Advances in Neural Information Processing Systems*, 2645–2653.

Robertson, N., and Seymour, P. D. 1984. Graph minors. iii. planar tree-width. *Journal of Combinatorial Theory, Series B* 36(1):49–64.

Szeliski, R.; Zabih, R.; Scharstein, D.; Veksler, O.; Kolmogorov, V.; Agarwala, A.; Tappen, M.; and Rother, C. 2008. A comparative study of energy minimization methods for markov random fields with smoothness-based priors. *PAMI* 30(6):1068–1080.

Tian, J.; He, R.; and Ram, L. 2012. Bayesian model averaging using the k-best bayesian network structures. *arXiv preprint arXiv:1203.3520*.

Wainwright, M. J., and Jordan, M. I. 2008. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning* 1(1-2):1–305.

Wainwright, M. J.; Jaakkola, T. S.; and Willsky, A. S. 2005. Map estimation via agreement on trees: message-passing and linear programming. *Information Theory, IEEE Transactions on* 51(11):3697–3717.

Yadollahpour, P.; Batra, D.; and Shakhnarovich, G. 2013. Discriminative re-ranking of diverse segmentations. *Proc. of IEEE Conference on CVPR*.

Yanover, C., and Weiss, Y. 2004. Finding the M most probable configurations using loopy belief propagation. In *Advances in Neural Information Processing Systems*.

Yen, J. Y. 1971. Finding the k shortest loopless paths in a network. *Management Science* 17(11):712–716.