Solving M-Modes in Loopy Graphs Using Tree Decompositions

Cong Chen CONG.CHEN@QC.CUNY.EDU
Changhe Yuan CHANGHE.YUAN@QC.CUNY.EDU
Ze Ye ZYE@GRADCENTER.CUNY.EDU
Chao Chen CHAO.CHEN@QC.CUNY.EDU
CUNY Graduate Center and CUNY Queens College

Abstract

M-Modes is the problem of finding the top M labelings of a graphical model that are locally optimal. The state-of-the-art M-Modes algorithm is a heuristic search method that finds global modes by incrementally concatenating MAP solutions in local neighborhoods. The search method also relies on the guidance of a heuristic function to explore the most promising parts of the search space. However, due to the difficulty of coordinating mode search, heuristic function calculation and local MAP computation in general loopy graphs, the method was only implemented and tested on special graphical models such as trees or submodular grid graphs. This paper provides a more general implementation of the search method based on tree decompositions that is applicable to general loopy graphs. A tree decomposition allows a sequence of local subgraphs to be mapped to a set of sub-trees 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. We use both random and real datasets to evaluate the effectiveness of the tree-decomposition method. Furthermore, we demonstrate the practical value of M-Modes in making multiple diverse structured predictions for a gesture recognition task.

Keywords: Graphical Model, Exact Inference, Heuristic Search, M-Modes

1. Introduction

In the classic structured prediction task, one computes a single solution of a given graphical model, i.e., the MAP. It has been shown that generating a set of M solutions that are both highly possible and diverse can be very useful in computer vision (Prasad et al., 2014) and computational biology (Fromer and Yanover, 2009). There are several popular approaches to the multiple-prediction problem. The classic M-Best methods (Dechter et al., 2012; Fromer and Globerson, 2009; Yanover and Weiss, 2004; Nilsson, 1998) computes the M most probable predictions. Solutions found by M-Best might be similar to each other and are not diverse. A recent method called Diverse M-Best (Prasad et al., 2014; Batra et al., 2012; Lampert, 2011) has been developed to compute candidates with both high probability and high diversity. Although the Diverse M-Best method shows impressive results in a number of computer vision applications, some researchers (Kirillov et al., 2015) argue that it suffers from its greedy nature—earlier solutions dominate ones found later.

M-Modes (Chen et al., 2013) is another approach for the multiple prediction problem that aims to find a set of top solutions that not only have a high probability but also are the MAPs in respective local neighborhoods. M-Modes is based on an inherent property of the distributional landscape description and is not biased by different search strategies.

Existing attempts for solving M-Modes (Chen et al., 2016, 2014, 2013) demonstrate that the M-Modes is a 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). It builds on the *Global-Local* property of modes, that is, a labeling is a *global* mode if and only if it is a *local* MAP in every connected δ-subgraph, where δ defines the size of local neighborhoods. The algorithm first computes all local MAPs of each subgraph conditional on different boundary configurations and then searches through all their consistent concatenations. The drawback of the DP algorithm is that most of its time is wasted on computing local MAPs that are never used in the global modes. A heuristic search approach has been developed to improve the efficiency of solving M-Modes (Chen et al., 2016). We will provide a more detailed review of this approach in a later section. Here it suffices to say that the search algorithms do not compute local MAPs in advance. In stead, it searches for a global mode by generating and verifying only necessary local MAPs on the fly.

The heuristic search method is *in principle* applicable to any graphical models. However, due to the difficulty of coordinating mode search, heuristic function calculation and local MAP computation in general loopy graphs, it was only implemented and tested on tree and submodular grid graphs. For chains or trees, MAP inference can be solved with simple belief propagation (Pearl, 1988). Submodular grid graphs satisfy the following properties (Boykov and Kolmogorov, 2004): 1) all variables have binary labels, and 2) parameters are all submodular, i.e., for each edge (u, v), $\theta_{uv}(1,1) + \theta_{uv}(2,2) \leqslant \theta_{uv}(1,2) + \theta_{uv}(2,1)$. For such grids, a *polynomial* min-cut algorithm can be used to compute local MAPs and heuristic functions (Chen et al., 2016).

This paper presents a more general implementation of the heuristic search method based on tree decompositions, which is applicable to general loopy graphical models. A tree decomposition allows us to choose an ordering of subgraphs such that their corresponding sub-tree representations sweep through the tree decomposition in an orderly fashion. This also allows the use of the Max-Product belief propagation (Wainwright et al., 2005) to be used for both computing local MAPs and for calculating heuristic functions. The heuristic functions are calculated *only once* in the reverse subgraph ordering and stored for repeated queries by the global modes search.

Note that the focus of this work is to propose a more general implementation of an existing method to overcome several technical difficulties presented by *general loopy* graphical models. In contrast, the existing implementation in (Chen et al., 2016) was tailored for special models such as trees and submodular graphs. Therefore, the new implementation, although more powerful, has worse efficiency or scalability than the existing implementation if forced to run on trees or submodular graphs.

2. Background

A discrete Markov random fields (MRF) (Kindermann and Snell, 1980; 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. Random variables correspond to vertices in the graph. The terms of variable and vertex are used interchangeably. A discrete value assigned to a variable is called a *label*. A label assignment for all variables is called a *labeling*. We use lowercase letters to denote labelings, such as x, and L_v to denote the label size for vertex v. We denote $f(x) = -\log(\prod_{c \in \mathcal{C}} \psi(x_c))$ as the energy of labeling x, where $\psi(x_c)$ is the potential function over a maximal clique x_c. The energy can be decomposed into a sum, i.e., $f(x) = \sum_{c \in \mathcal{C}} f_c(x_c)$, in which $f_c(x_c) = -\log(\psi(x_c))$. The energy is proportional to the negative log probability. Thus, finding a solution with the maximal probability is equivalent to minimizing the energy.

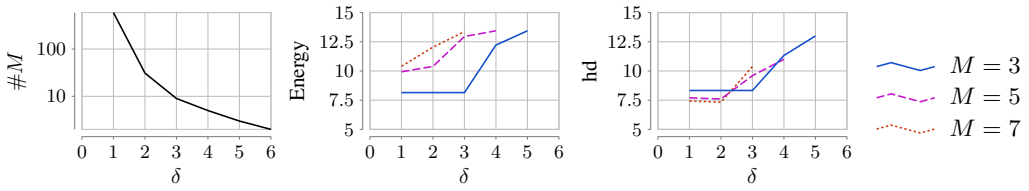

Figure 1: The effect of δ on (Left) total number of modes, (Middle) the energy of the M-th mode (negatively proportional to log probability), and (Right) the average pairwise hamming distance on dataset Child.

2.1 M-Modes

We define the precedence of a labeling x over another labeling y, i.e., $x \prec y$, as either (1) the energy of x is less than that of y, or (2) they have the same energy, and x is smaller than y in lexicographical order. We use lexicographical order as a tie breaker to ensure that there is at most one mode within each given δ-neighborhood. We say $x \prec \mathcal{X}$ when a labeling x has the highest precedence in the set $\mathcal{X}$ (x precedes all the other labelings in set $\mathcal{X}$).

We use the *Hamming distance* $\mathrm{hd}(\cdot, \cdot)$, i.e., the number of disagreed variables between two labelings of equal length, as the distance metric. Given a non-negative integer δ, called *scale*, the *δ-neighborhood* of x is defined as $\mathcal{N}_\delta(x) \triangleq \{\, x' \mid \mathrm{hd}(x, x') \leqslant \delta \,\}$. We define δ-modes as follows.

Definition 1 (δ-Mode) *x is a δ-mode if and only if $x \prec \mathcal{N}_\delta(x)$.*

A δ-mode labeling is defined to precede all other labelings in its δ-neighborhood. This ensures the δ-modes are diverse; any two modes are at least δ away. The number of modes typically decreases exponentially as δ increases. A larger δ increases the diversity of the solutions. 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, δ provides a tradeoff between the diversity and probability of modes. Figure 1 provides visual illustrations of the effect of δ on a UCI dataset called Child. Finally, we define the main computational problem as follows. For simplicity, we drop δ from the notations hereafter.

Problem 1 (M-Modes) *Given a graphical model and a scale δ, compute the top M modes.*

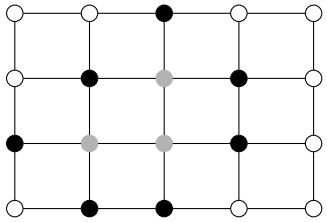

Figure 2: An example of a δ-subgraph with $\delta = 3$. The gray vertices are the interiors and the black vertices are the boundaries. Others are exteriors.

2.2 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, all variables that are adjacent to variables in S are its *boundaries*, denoted as ∂S. Denote by $cl(S)$ the disjoint union of S and its boundaries, $S \cup \partial S$. All variables outside $cl(S)$ are *exteriors*. For convenience, we also call the variables in S *interiors*. Figure 2 shows a δ-subgraph with $\delta = 3$ in a grid graph. The interiors, boundaries and exteriors are colored gray, black and white respectively.

A label assignment to a δ-subgraph, S, is called a *local labeling*. Given a label assignment to boundaries of S, ∂S, the highest precedential local labeling of S is called a *local MAP*. Again consider Figure 2, fixing the labels of boundaries (black) can uniquely determine the local MAP of interiors (gray).

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, 2016). Formally, we have:

Theorem 2 (Global-Local) *A labeling is a δ-mode $\Longleftrightarrow$ its local labelings of all δ-subgraphs are local MAPs.*

3. Solving M-Modes using Tree Decompositions

The goal of this paper is to develop a general implementation of the heuristic search method for solving M-Modes (Chen et al., 2016) to be applicable to general loopy graphs. The main idea is to use tree decompositions of the models to coordinate various operations needed for the M-Modes search. We first outline the major steps of the existing M-Modes search algorithms (Chen et al., 2016). We then review the basics of tree decomposition. After that, we explain how the tree decompositions can be used to coordinate cluster and vertex orderings, calculate heuristic functions, map δ-subgraphs to minimum sub junction trees, and compute local MAPs. Finally, we integrate all these techniques into a tree-decomposition A* search algorithm.

3.1 M-Modes Search

The heuristic search algorithms for solving M-Modes proposed in (Chen et al., 2016), including depth first branch and bound (DFBnB), and A star (A*), are developed based on Theorem 2 and work as follows.

First, we create a list of all the δ-subgraphs of a graphical model. Given a contiguous ordering of the vertices in a graphical model, the δ-subgraphs are created by recursively adding adjacent vertices such that adjacent subgraphs are maximally overlapped to facilitate concatenating consistent local MAPs.

Second, MAP inference in the reverse order of the δ-subgraph ordering is used to calculate a heuristic function. The heuristic function only needs to be calculated once and is stored for repeated queries by the global mode search.

Third, we search for top modes as follows. The search tree is initialized with a root node representing an empty labeling. The precomputed δ-subgraphs are sequentially used to expand the search tree. At each step, conditioning on a current *frontier* search node representing a partial labeling of all variables, the next δ-subgraph must have all interiors and some boundaries already fixed, but other boundary variables remain unknown. For each labeling of the unknown boundaries, the local MAP over the interiors is calculated. If the local MAP is consistent with the known values of the interiors, a new successor node is created with a new partial labeling consisting of parent's partial labeling plus labeling of new boundaries. The new labeling is used to look up a heuristic value from the stored heuristic function. The total score of the partial labeling plus heuristic value is used to perform A* or DFBnB search. The search continues until M solutions are found by A* or the search tree is exhausted by DFBnB.

The advantage of the search algorithms over dynamic programming (Chen et al., 2013, 2014) is clear: They only compute local MAPs that are needed during the search, and the heuristic functions guide the search to only explore the most auspicious search space. However, these existing M-Modes search algorithms are not flexible enough to deal with general loopy graphs. We propose to use tree decompositions to generalize these methods.

3.2 Tree Decompositions

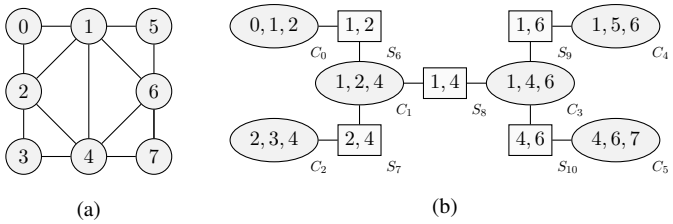

Figure 3: Example of Tree Decomposition: (a) the original graphical model and (b) the junction tree. The ellipses are clusters, and the rectangles are sepsets.

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*. 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 satisfies two properties: (1) *running intersection* property: given two clusters X and Y, all clusters on the path between them contain $X \cap Y$, and (2) Markov property: any cluster d-separates its adjacent clusters. Figure 3 shows an example of tree decomposition.

The junction tree encodes a joint probability distribution of the labeling for the graph according to: $p(x) = \frac{\prod_i p(C_i)}{\prod_j p(S_j)}$, where $p(C_i)$ and $p(S_j)$ are joint probability distributions for cluster C_i and sepset S_j respectively. And for each cluster C and neighboring sepset S, it holds that $p(S) = \sum_{C \setminus S} p(C)$. As a sepset is a subset of its neighbor cluster, its probability marginalizes the cluster probabilities. Besides, it also holds that $p(X') = \sum_{X \setminus X'} p(X)$, where $X' \subset X$, marginalizing all the other vertices in the same cluster.

Therefore, let $f(C_i) = -\log(p(C_i))$ and $f(S_i) = \log(p(S_i))$ (notice: removed negative), we define the energy function of x as $f(x) = \sum_i f(X_i)$, where X_i is the local labeling for each cluster. We can do MAP inference using Max-Product belief propagation (Wainwright and Jordan, 2008) to seek the minimum energy. Roughly speaking, the algorithm sums over branches at the clusters, and performs minimizations at the sepsets; The minimizations take place when cluster energies are being propagated to its neighboring sepsets. This algorithm is crucial and fundamental for local MAP calculations used in the proposed M-Modes algorithm.

3.3 Select Cluster and Vertex Orderings

The first key step of existing M-Modes search is selecting the ordering of all δ-subgraphs, which were enumerated according to a given contiguous ordering of variables in the graphical model. The δ-subgraphs ordering is important because it affects both the ordering of the search steps and the calculation of a heuristic function.

To work with tree decompositions, we have two orderings that have to be created in coordination: a *cluster ordering* of the junction tree and a *vertex ordering* of the original graph. Given a tree decomposition, we create the cluster ordering by picking any cluster as the root and performing a depth-first traversal. After that, we create a vertex ordering by examining the clusters sequentially and wait until the last chance to add a vertex to the vertex ordering. Consider Figure 3. Suppose we have created the following cluster ordering: C_0, C_1, C_2, C_3, C_4 and C_5. Vertices 0, 1 and 2 belong to cluster C_0. Vertex 0 does not appear in later clusters, so this is the last chance to add vertex 0 to the vertex ordering. However, vertices 1 and 2 appear in later clusters; we do not add them just yet. We do the same for all subsequent clusters until a full vertex ordering is created. Algorithm 1 is a pseudo code for the strategy. Inside, the $\mathcal{C}_{\{v\}}$ means a set of clusters which vertex v belongs to. However, note that the strategy for selecting when to add a vertex is not unique. For example, we can also add a vertex when it first appears.

Algorithm 1 Finding consistent vertices ordering with clusters ordering

Input: $\vec{\mathcal{C}}$ – ordered cluster list;
$\mathcal{V}$ – unordered vertices set
Output: $\vec{\mathcal{V}}$ – ordered vertices list
function CONSIST-ORDERING($\vec{\mathcal{C}}$)
for $\forall C \in \vec{\mathcal{C}}$ **do**
for $\forall v \in C$ **do** $\mathcal{C}_{\{v\}}$.Add(C)
for $\forall C \in \vec{\mathcal{C}}$ **do**
for $\forall v \in \mathcal{V}$ **do**
if $C \in \mathcal{C}_{\{v\}}$ **then** $\mathcal{C}_{\{v\}}$.Remove(C)
if $\mathcal{C}_{\{v\}}$.Is-Empty() **then**
$\vec{\mathcal{V}}$.Append(v)
return $\vec{\mathcal{V}}$

3.4 Minimum Sub Junction Trees

After getting the vertex ordering, we create a list of δ-subgraphs using the method described in Section 3.1 (Chen et al., 2016). The δ-subgraphs are the basic search units of the existing M-Modes search algorithm. In order 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) the δ-subgraph spans as a *sub junction tree*:

Definition 3 (Sub Junction Tree) *For δ-subgraph S, a Sub junction tree T_S is a sub tree of a junction tree T that includes all the vertices in $cl(S)$.*

Sepsets are always subsets of its adjacent clusters in a junction tree. In Figure 3, assume we want to identify the sub junction tree for 1-subgraph of vertex 0. The interior is vertex 0, and the boundaries are vertices 1 and 2. So, a corresponding sub junction tree can either include just the cluster C_0 or include C_0, S_6, and C_1. Either is serviceable. However, the latter sub junction tree has a cluster containing an extra vertex 4 that needs to be marginalized out. In order not to waste computation, we should find a *minimum sub junction tree* (MSJT) that adequately represents a δ-subgraph for a local MAP calculations:

Definition 4 (Minimum Sub Junction Tree (MSJT)) *$\hat{T}_S$ is a MSJT if it is a sub junction tree and consists of fewest clusters.*

There is only one such MSJT (uniqueness of MSJT) for each δ-subgraph, i.e., removing one more cluster does not form an adequate sub junction tree, and adding one more cluster makes it not minimum. In that example, the cluster C_0 is the MSJT consisting of just one cluster and containing all relevant vertices. Then, how can we quickly find such an MSJT? Theorem 5 claims

that collecting the clusters which contains the interiors is sufficient to obtain an MSJT, which would automatically include the boundaries.

Theorem 5 (Finding MSJT) *$\hat{T}_S$ is an MSJT $\Leftrightarrow$ $\hat{T}$ satisfies: (1) $\forall v \in S$, $\exists X \in \hat{T}$, $v \in X$, and (2) $\forall X \in \hat{T}$, $\exists v \in X$, $v \in S$. X are clusters of $\hat{T}_S$.*

Proof (1) essentially says that each interior of δ-subgraph S must belong to some cluster in the MSJT, which is trivially true. (2) claims that at least one interior should be in each cluster in the MSJT. In other words, there is no cluster that only contains boundaries and maybe some other exteriors, called boundaries clusters, in an MSJT. Otherwise, there must exist a neighboring cluster that shares a sepset and the sepset only contains boundaries, which defies the need to include the boundaries cluster. ■

Therefore, an algorithm for getting an MSJT $\hat{T}_S$ for a δ-subgraph S works as follows. First, starting from an empty set T, we enumerate all the interiors $v \in S$. For each interior, we get all the clusters containing it and insert them into the set. In the end, the set T is the MSJT for S.

3.5 Calculating local MAP in an MSJT

Each step of M-Modes search needs to perform a local MAP inference. We now discuss how to perform the MAP inference using MSJTs. Because of the need to distinguish interiors and boundaries as well as exteriors, we have to tailor the Max-Product belief propagation algorithm to achieve that. There are three possible scenarios in one cluster: (1) all vertices are interiors, (2) there are both interiors and boundaries, or (3) there is a mixture of interiors, boundaries and exteriors.

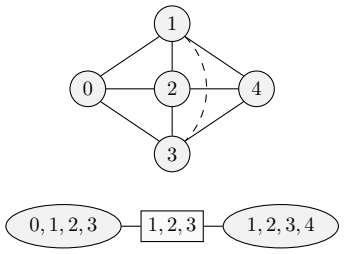

Figure 4: Example of clusters with mixture of interiors, boundaries and exteriors: (Top) the original graph and (Bottom) the junction tree, which introduces an edge from vertex 1 to vertex 3, so that $1 \not\perp 3 \mid 0, 2, 4$ after tree decomposition. And for given interiors 1 and boundaries $0, 2, 4$, there is an exterior 3 in the clusters.

For scenario (1): Since all of the vertices are interiors, we just enumerate all energies in the cluster. For scenario (2): there are some boundaries which are fixed, so that we skip those energies whose labels disagree with the labels of the boundaries. Again consider Figure 3, assuming in cluster C_0, the interior is vertex 0, the boundaries are vertices 1 and 2 with labels 2 and 2, and the label size for all of them are 3. Therefore, we merely itemize three δ-labelings which are 022, 122, and 222.

Scenario (3) is the trickiest situation. First let us look at an example in Figure 4. After building a tree decomposition, we lost the d-separation between vertices 1 and 3. It results in a situation where a cluster could contain interiors, boundaries and exteriors altogether. Accordingly, the correct energy list involves not only fixing boundaries, but also marginalizing exteriors by fixing them to an arbitrary value (because the dummy connection has no real effect). For efficiency, this marginalization can be pre-calculated and stored in the energy list of each cluster for subsequent usage.

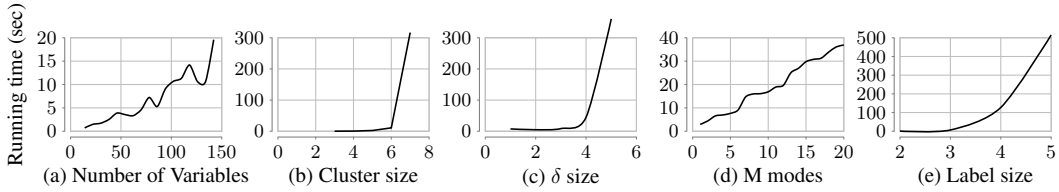

Figure 5: Experimental results for A* on synthetic binary tree decomposition models of different property settings

3.6 Tree Decomposition M-Modes Search

Finally, we integrate all the techniques in the previous sections into a new implementation of the A* algorithm that uses tree decompositions of graphical models to solve M-Modes. The algorithm is outlined in Algorithm 2. The major differences are as follows. Instead of δ-subgraphs, the new implementation takes a list of MSJTs as input. The heuristic function is calculated in the reverse order of the cluster ordering. The search layers correspond to different MSJTs. At each search step, it is necessary to perform a local MAP inference at the corresponding MSJT.

Algorithm 2 Tree decomposition M-Modes search

Input: a junction tree; δ size; number of modes
Output: a set of modes

1. Get a cluster ordering of the clusters $\vec{\mathcal{C}}$ (DFS)
2. Create vertex ordering $\vec{\mathcal{V}}$ with $\vec{\mathcal{C}}$ (see Alg. 1)
3. Find δ-subgraph list $\vec{\mathcal{S}}_\delta$ by $\vec{\mathcal{V}}$ (Chen et al., 2016)
4. Get $\vec{\mathcal{T}}$ s.t. $\forall S_\delta \in \vec{\mathcal{S}}_\delta$, MSJT $T_S \in \vec{\mathcal{T}}$ (see Thm. 5)
5. Use Max-Product algorithm on the $\vec{\mathcal{C}}$.reverse() to calculate heuristic functions
6. Perform A* M-Modes search over $\vec{\mathcal{T}}_S$

4. Computational Complexity of M-Modes

We briefly discuss the computational complexity of M-Modes inference. Consider a graphical model with n variables and label size l. The number of all δ-subgraphs is $\mathcal{O}(\binom{n}{\delta})$, where $\binom{n}{\delta}$ is the binomial coefficient. However, only δ-subgraphs with new boundaries result in branching layers in the search tree, while others lead to verification layers (no branching). The maximum number of branching layers is n, so the number of branching nodes is $\mathcal{O}(l^n)$. The number of verification layers is $\mathcal{O}(\binom{n}{\delta})$, resulting in $\mathcal{O}(l^n\binom{n}{\delta})$ verification nodes. Each branching or verification node need a local MAP inference, which is $\mathcal{O}(l^\delta)$. Therefore, the total complexity is $\mathcal{O}(l^{\delta+n}\binom{n}{\delta})$.

Tree decomposition indicates the presence of conditional independence in a graphical model and can effectively reduce the number of δ-subgraphs, and hence computational complexity. Let the maximum size of clusters be c, number of clusters be k, and the maximum number of clusters that a δ-subgraph can span be s. The number of all δ-subgraphs is now $\mathcal{O}(\binom{k-1}{s-1}\binom{cs}{\delta})$. The total complexity is reduced to $\mathcal{O}(l^{\delta+n}\binom{k-1}{s-1}\binom{cs}{\delta})$. In any case, M-Modes is believed to be much more difficult than M-Bests, which aims to find M top labelings with highest probabilities.

5. Experiments

The existing A* implementation in (Chen et al., 2016) was tailored for trees and submodular graphs. In comparison, the new implementation targets general loopy graphs. Consequently, it has worse efficiency and cannot scale to the large submodular graphs tested in (Chen et al., 2016). We therefore focus on evaluating the new implementation on *general loopy* graphs. The experiments were

performed on an IBM System with 32 core 2.67GHz Intel Xeon Processors and 512G RAM. And the program is written in language C++ using the GNU compiler G++ on a Linux system.

5.1 Synthetic Models

We first generated random tree decompositions with different settings to test the impact of various properties of the decompositions on M-Modes. We generate a tree decomposition as follows: We start by creating a root cluster with certain size, randomly select a number vertices from the root as separator, and create another cluster with the same size sharing the separator. Then, we randomly pick an existing cluster and create A neighboring cluster in the same way until we create enough clusters. Last, we add random potentials to each cluster. We want to test five properties, including the number of variables, cluster size, δ size, size of M, and label size. Each time we fixed four properties and varied the remaining one. The base setting has the number of variables being around 100, the cluster size being 6, δ size being 3, M being 4 and label size being 3. For each setting, we generated ten different models and computed the median running time. See Figure 5 for empirical results of synthetic models.

The results show that all of the properties affect the running time of M-Modes. However, several of the properties have more significant impact than others. In particular, cluster size, δ and label size seem to impact the running time exponentially. The cluster size and δ directly impact the number of possible δ-subgraphs. The label size affects the number of label combinations and, hence, size of search space. It is quite understandable why they have an exponential impact on running time M-Modes. In comparison, The number of modes, M, only has a linear impact because the cost to obtain each mode is roughly the same. Finally, the impact of number of variables is less clear and has a much higher variance. We offer the following explanation. Even though the number of variables directly affects the number of δ-subgraphs, but because the cluster size is fixed, so the increase in running time is somewhat constrained. The empirical observations are mostly consistent with our analysis of the computational complexity of M-Modes.

5.2 Benchmark Models

Name	N	L	Cl	Sub	Time
Adult	15	3	5	59	0.93
Child	20	6	4	92	1.57
Flag	29	3	7	133	87.69
Alarm	37	4	5	177	0.20
Spectf	45	2	9	190	380.15

Figure 6: Running time of A* on benchmark models (sec): **N** is the number of variables, **L** is maximum label size, **Cl** is maximum cluster size, **Sub** is number of δ-subgraphs, and **Time** is the running time (sec).

We also tested A* on several benchmark models that are created from either benchmark Bayesian networks, including Child (Spiegelhalter and Cowell, 1992) and Alarm (Beinlich et al., 1989), or learned from UCI datasets. Bayesian networks are first moralized into undirected Markov networks. We then create tree decompositions for the models using greedy heuristic such as min-fill.

In these experiments, we set $\delta = 3$ and $M = 4$. Figure 6 show the performance of A*. We also list some important properties of the benchmark models: the number of variables, maximum label size, maximum cluster size, and number of δ-subgraphs.

The results are mostly consistent with what we have observed on the synthetic datasets. The two most difficult models are Flag and Spectf. They are difficult because they have relatively large maximum cluster

size. Even though Alarm has a large number of δ-subgraphs, its tree decomposition has smaller cluster sizes than Flag, indicating smaller branching factor. As a result, it is still a easy model to solve for A*.

5.3 Diverse Structured Prediction

Finally, we demonstrate the practical value of M-Modes in a gesture recognition task and show that modes can significantly outperform the state-of-the-art diverse structured prediction method called Diverse M-Best (Batra et al., 2012). We use the gesture recognition dataset from ChaLearn challenge (Guyon et al., 2011). This dataset consists of 20 video batches (devel 01–20), each of which includes 47 RGB video sequences and 47 corresponding depth videos recorded with the Kinect camera. In each video sequence, one actor made 1–5 gestures drawn from 8 to 15 gesture vocabularies. See Figure 7(Top) for example frames. The local features employed were HOG and HOF descriptors from both RGB and depth images, based on STIP detector. Finally, each video segment of 30 frame-length was represented by a 60-dimensional bag-of-word (BOW) feature vector. Within each batch, the first 30 video sequences was used for training and the rest for testing. We train a Conditional Random Field (CRF) (Lafferty et al., 2001) for the task. The graphical model has tree-width three; each node (corresponding to a 30 frames video segment) is connected to the two nodes before it and the two nodes after it. See Figure 7(Middle). Each sample ranges from 9 to 60 nodes with average 27.

Following the tradition of other diverse prediction problem, we let the methods predict up to M solutions and evaluate those by their *oracle accuracy*, i.e., the highest accuracy of one of the M solutions compared with the ground truth. The task of selecting one labeling out of M candidates is beyond the scope of this paper. The accuracy of a solution is measured as the number of correct labels normalized by the total number of nodes of the graph (one minus the normalized Hamming distance between a labeling and the ground truth). Each method has its own parameter (the parameter for M-Modes is δ). We select the optimal parameter for each method using the first batch, and then evaluate the accuracy on the remaining 19 batches. Note that for different M the optimal parameter value are different.

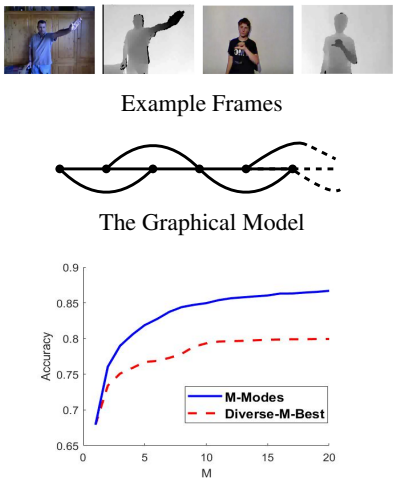

Figure 7: Comparison of prediction methods

The results at different M are shown in Figure 7(Bottom). M-Modes performs significantly better than Diverse M-Best; at $M = 3$, M-Modes reaches a much higher accuracy than Diverse M-Best. The accuracy of both M-Modes and Diverse M-Best monotonically increases as M increases, because both methods propose monotonically growing set of solutions as M increases.

6. Conclusions

We have developed a new implementation of the M-Modes search method (Chen et al., 2016) based on tree decompositions in order to be applicable in general loopy graphical models. A tree decomposition allows us to map the δ-subgraphs of a graphical model to their respective minimum

sub junction trees on the decomposition. Such a mapping further allows easy coordination between three things that are critical in solving M-Modes: 1) creating a sequence of all δ-subgraphs, 2) computing local MAPs, and 3) calculating heuristics for each search step. We have evaluated the proposed methods with A* on a set of synthetic and benchmark datasets. The results demonstrate that various properties of a graphical models affect the difficulty of the M-Modes problems. Among them, the induced widths of tree decompositions, the δ size, and the label size seem to have the largest impact.

Although promising, the proposed M-Modes methods are shown to only scale to relatively small graphical models with just dozens of variables, laughable compared to other inference problems such as MAP. Although heuristic functions already help A* tremendously in achieving better practical performance, the inherent difficulty of solving M-Modes makes larger graphical models out of reach, still.

We believe, however, our proposed methods are the first tries. More advanced M-Modes are yet to be developed. For example, it is well known that formulating a search space as a graph rather than a tree usually brings tremendous improvements. Also, tree decomposition is a way to utilize the conditional independence relations present in a graphical model, but more sophisticated approaches for exploring such independence relations can further help.

Acknowledgements

This research is partially supported by the NSF grants IIS 1718802, IIS 1829560, and CCF 1733866.

References

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

I. A. Beinlich, H. J. Suermondt, R. M. Chavez, and G. F. Cooper. *The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks*. Springer, 1989.

Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(9):1124–1137, 2004.

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

C. Chen, H. Liu, D. Metaxas, and T. Zhao. Mode estimation for high dimensional discrete tree graphical models. In *Advances in neural information processing systems*, pages 1323–1331, 2014.

C. Chen, C. Yuan, and C. Chen. Solving m-modes using heuristic search. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI-16)*, New York, NY, 2016.

R. Dechter, N. Flerova, and R. Marinescu. Search algorithms for m best solutions for graphical models. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, AAAI'12, pages 1895–1901. AAAI Press, 2012. URL `http://dl.acm.org/citation.cfm?id=2900929.2900996`.

M. Fromer and A. Globerson. An LP view of the M-best MAP problem. *Advances in Neural Information Processing Systems*, 22:567–575, 2009.

M. Fromer and C. Yanover. 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, 2009.

I. Guyon, V. Athitsos, P. Jangyodsuk, and H.-J. Escalante. ChaLearn gesture challenge, 2011. https://sites.google.com/a/chalearn.org/gesturechallenge/2011-one-shot-learning, [Accessed: Nov. 2012].

R. Kindermann and J. L. Snell. *Markov random fields and their applications*, volume 1. American Mathematical Society, 1980.

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

J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML)*, pages 282–289, 2001. ISBN 1-55860-778-1.

C. Lampert. Maximum margin multi-label structured prediction. *NIPS*, 2011.

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

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

J. Pearl. 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, 1988.

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

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

D. J. Spiegelhalter and R. G. Cowell. Learning in probabilistic expert systems. *Bayesian statistics*, 4:447–465, 1992.

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

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

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