Efficient Heuristic Search for M-Modes Inference

Cong Chen CONG.CHEN@QC.CUNY.EDU
Changhe Yuan CHANGHE.YUAN@QC.CUNY.EDU
Chao Chen CHAO.CHEN.CCHEN@GMAIL.COM
The City University of New York and Stony Brook University

Abstract

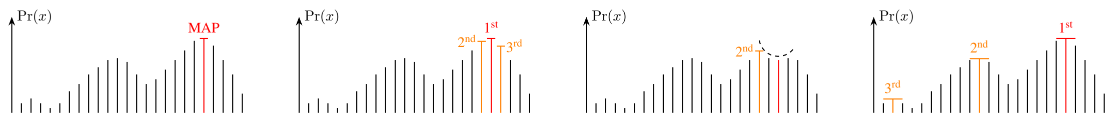
M-Modes is the problem of finding the top M locally optimal solutions of a graphical model, called modes. These modes provide geometric characterization of the energy landscape of a graphical model and lead to high quality solutions in structured prediction. It has been shown that any mode must be a local MAP within every subgraph of certain size. The state-of-the-art method is a search algorithm that explores subgraphs in a fixed ordering, uses each subgraph as a layer and searches for a consistent concatenation of local MAPs. We observe that for the M-Modes problem, different search orderings can lead to search spaces with dramatically different sizes, resulting in huge differences in performance. We formalize a metric measuring the quality of different orderings. We then formulate finding an optimized ordering as a shortest path problem, and introduce pruning criteria to speed up the search. Our empirical results show that using optimized orderings improves the efficiency of M-Modes search by up to orders of magnitude.

Keywords: Graphical Model; Exact Inference; Heuristic Search; M-Modes.

1. Introduction

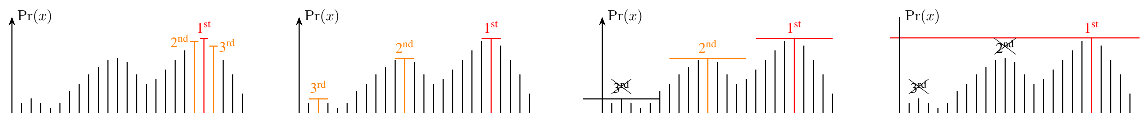
For inference in probabilistic graphical model, much effort has been directed at algorithms for obtaining a single solution with the highest probability (MAP) (Figure 1(a)). In many applications, it is useful to find a set of top solutions, i.e., M-Best (Nilsson, 1998; Dechter et al., 2012). However, the M most probable configurations tend to be very similar to the MAP solution or to each other, thus lacking diversity (Figure 1(b)).

Diverse multiple inference tries to address the drawback by attempting to find a set of solutions with both high probability and high diversity and has shown impressive results in a number of computer vision (Batra et al., 2012; Yadollahpour et al., 2013; Kirillov et al., 2015), computational biology (Fromer and Yanover, 2009), and machine translation (Gimpel et al., 2013) where multiple hypotheses are preferred for advanced reasoning. One method called Diverse M-Best (Batra et al., 2012) finds multiple diverse solutions one by one greedily such that each solution has the highest probability among all solutions that are certain distance away from existing solutions (Figure 1(c)).



(a) MAP (b) M-Best (c) Diverse M-Best (d) M-Modes

Figure 1: Illustration of four inference problems. Each vertical bar corresponds to a labeling. The height corresponds to its probability.



(a) $\delta = 0$ (b) $\delta = 1$ (c) $\delta = 4$ (d) $\delta = \infty$

Figure 2: 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.

M-Modes (Chen et al., 2013), as another multiple inference method, aims to find a set of top solutions that not only have high probabilities 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 (Figure 1(d)). The state-of-the-art algorithm for solving M-Modes is a heuristic search approach based on tree decompositions that is applicable to loopy graphs (Chen et al., 2018).

In this work, we propose an improvement of current heuristic search method by optimizing a better subgraph ordering. The key observation is that different orderings of subgraphs can lead to very different search spaces. Only newly included boundary variables called *frontiers* will introduce new branches to the search tree; other variables all correspond to verification layers without branching. Therefore, an ordering with fewer frontiers can reduce the *total size* of the search space exponentially. Based on the observation, we define a metric for evaluating the quality of search orderings. We then formulate finding an optimized ordering as a shortest path problem. Besides, we define an interesting related δ-vertex cover graph theory problem and prove its NP-completeness. Empirical results show that using optimized orderings improve the efficiency of M-Modes search by up to orders of magnitude.

2. Background

2.1 M-Modes Problem

The *M-Modes* 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.

Given a non-negative integer δ, δ-neighborhood $\mathcal{N}_\delta(y)$ is defined as $\mathcal{N}_\delta(y) = \{\, x \mid \Delta(x, y) \leqslant \delta \,\}$. Without loss of generality, we assume using Hamming distance. Thus, a labeling y is a δ-mode as y has highest probability (lowest energy) in its δ-neighborhood. So, M-Modes is a multiple inference 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. If δ is set too large, too many high-probability solutions are suppressed by superior neighbors, and the top modes may contain too many low-probability solutions. Thus, the δ provides a tradeoff between the diversity and probability of modes. Figure 2 illustrates what mode solutions and δ-neighborhoods are and how the scale δ affects the number of modes. When $\delta = 0$, every labeling is a mode so that the problem becomes M-Best; when $\delta = \infty$, the MAP solution is the only mode.

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_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*.

A label assignment to a δ-subgraph, S, is called a *local labeling*. Given a label assignment to boundaries of S, the highest precedential local labeling of S is called a *local MAP*. We say two local labelings are consistent if they agree on the overlaps. For example,$\langle 101\ldots\rangle$ and $\langle \_011\ldots\rangle$ are consistent 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 used by several recent algorithms for solving M-Modes (Chen et al., 2016, 2018). 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.*

In order to use the Global-Local Theorem to find 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 of a given graph. Table 1 shows the number of δ-subgraphs of different δ size ($|\mathcal{S}_\delta|$) on three benchmark datasets. It empirically shows δ-subgraph set sizes are about exponentially increasing to δ size (when δ is much smaller than the number of variables).

$\delta =$	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

Table 1: δ-subgraph set sizes on dataset Child, Alarm, and Hepar2

2.3 Solving M-Modes

The computation of M-Modes is challenging. Based on this Global-Local Theorem, a dynamic programming algorithm (Chen et al., 2013, 2014) for solving M-Modes, first computes all local MAPs of each subgraph conditional on different boundary configurations. Next, it searches through all the consistent concatenations of these local MAPs. The computational bottleneck is spending most of the time computing unnecessary local MAPs; many of them are never used in any mode due to inconsistency.

The current algorithm for solving M-Modes is a heuristic search approach which generates and verifies necessary local MAPs *on the fly* (Chen et al., 2016). The algorithm explores all subgraphs one-by-one according to a given ordering. At each step, local MAPs of a subgraph are computed for different boundary variable values, and are used to verify that only permissible successor search nodes are generated. The advantage of the search algorithm is that it computes only local MAPs that are needed during the search, and heuristic functions guide the search to explore only the most promising 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. Thus, Chen et al. (2018) provides a more general implementation of the search method based on tree decompositions that is applicable to general loopy graphs. See Table 2 for listing the evolution of algorithms for solving M-Modes.

Chains	**Trees**	**Loopy Graphs**
DP (Chen et al., 2013)	DP (Chen et al., 2014)	–
–	HS* (Chen et al., 2016)	HS + TD (Chen et al., 2018)

Table 2: Evolution of algorithms solving M-Modes on chains, trees, and general loopy graphs. DP: Dynamic Programming; HS: Heuristic Search; TD: Tree Decomposition. * HS can also solve submodular grid graphs;

3. Optimizing Heuristic Search Ordering for Solving M-Modes

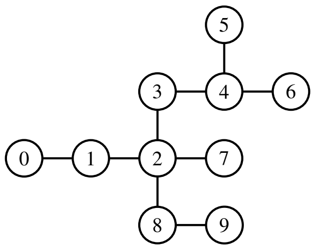


Figure 3: A tree graph

In this section, we 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.

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 as our example for easier illustration; the same idea easily generalizes to loopy graphs by tree decomposition. 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(a)). Then, for each δ-subgraphs, we circumscribe it and get boundary variables (See Column $\partial\mathcal{S}_\delta$ in Table 3(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 MAP for the interior variable $\{2\}$.

We first perform M-Modes search using the δ-subgraph ordering $\mathcal{S}_\delta^+[\vec{ord}_1]$ in Table 3(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. Afterwards, 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 a 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; 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.

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)

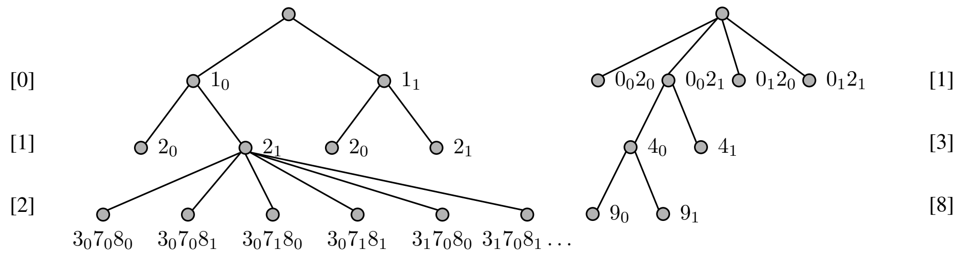
Table 3: δ-subgraphs for the tree graph in Figure 3 where $\delta = 1$; (a) **Index** are the indices assigned to all δ-subgraphs, $\mathcal{S}_\delta$ are the interior variables of each δ-subgraphs, $\partial\mathcal{S}_\delta$ are the boundary variables; (b) $\mathcal{S}_\delta^+[\vec{ord}_1]$ are the frontiers given the default (naive) ordering $\vec{ord}_1$; (c) $\mathcal{S}_\delta^+[\vec{ord}_2]$ are the frontiers given an optimized ordering $\vec{ord}_2$.

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 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 4(a) for the partial search tree created above.

A Better Search Ordering: Now consider another δ-subgraph ordering in Table 3(c). We start by searching the δ-subgraph with interior 1 and boundaries $\{0, 2\}$. Up to four search nodes corresponding 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 to search 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 to search 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 need 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 a much smaller search space. See Figure 4(b) for part of the new search tree.

3.1 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



(a) Default Ordering (b) Optimized Ordering

Figure 4: Two example search trees (first three layers) for solving M-Modes for the tree graph in Figure 3 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.

nodes is exponential in the number of new boundary variables. We call these undecided boundaries *frontiers*.

Definition 1 (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(b) and 3(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(b) introduced 8 frontiers in total, while ord_2 in Table 3(c) introduced only 4 frontiers in total.

3.2 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:

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

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:

$$\left|\mathrm{Leaf}(\mathcal{S}_\delta[\vec{ord}])\right| = \prod_{i=0}^{\left|\mathcal{S}_\delta^+[\vec{ord}]\right|-1} \prod_{v \in S_\delta^+[i]} L_v = \prod_{v \in \mathcal{S}_\delta^+[\vec{ord}]} L_v \tag{2}$$

3.3 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.*

$$\arg\min_{\vec{ord}} \sum_{v \in \mathcal{S}^+_\delta[\vec{ord}]} \log(L_v) \tag{3}$$

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 search step makes sure that no gap is introduced between δ-graphs in the (partial) ordering throughout the search. Once 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.

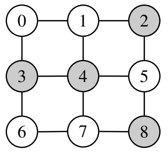
Finding a search ordering is only a preprocessing step for the actual M-Modes search. We cannot afford to spend too much time here. If the number of δ-subgraphs is large, solving the shortest-path problem may take a long time, because there are $|\mathcal{S}_\delta|\,!$ possible solutions. In the following 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.

4. δ-Vertex Cover Problem

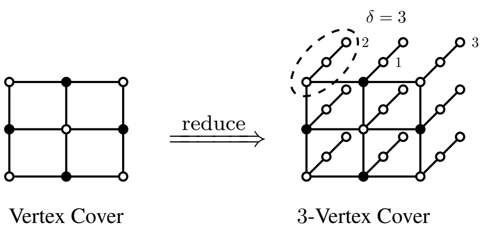
Ignoring label size for simplicity[1], Optimizing search orderings for M-Modes is equivalent to solving a problem we call δ-vertex cover, formally define as follows.

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 $v \in \mathcal{V}'$.*

1. The case considering label sizes is related to the *weighted* vertex cover problem.



(a)



(b)

Figure 5: (a) A 2-Vertex Cover Example; (b) An example of reducing a vertex cover problem to a 3-vertex cover problem. The black vertices are the covered vertices, the solutions to each problems.

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 a 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.

Example. *Retrieving an optimized search ordering from a given δ-vertex cover result:*

Assume we already have a δ-vertex cover result, which $\delta = 2$, for a 3×3 grid graph. See Figure 5(a). It is trivial to know that each edge belongs to a 2-subgraph's interiors.

There are 4 vertices covered: 2, 3, 4, and 8. And connected subgraphs decomposed by the vertex cover are $\{0, 1\}$, $\{5\}$, and $\{6, 7\}$, none of which is larger than 2.

In order to retrieve a search ordering, we can start by creating a 2-subgraph containing $\{0, 1\}$; Its boundaries are $\{2, 3, 4\}$. Up to now we have visited $\{0, 1, 2, 3, 4\}$. Next we go to 5. Because 5 is only 1 vertices, we need to include another visited vertex such that the adding will not introduce vertices not in the cover set. So, we have to choose 2, forms subgraph $\{2, 5\}$. Now we have visited $\{0, 1, 2, 3, 4, 5, 8\}$. The last step is trivial, that we just add subgraph $\{6, 7\}$ to finish the work.

Therefore, based on the given δ-vertex cover, the subgraph ordering we get is $\{0, 1\} \rightarrow \{2, 5\} \rightarrow \{6, 7\}$.

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 prove the latter is also NP-complete.

Theorem 2 (δ-Vertex Cover Complexity). *The δ-vertex cover problem is NP-complete.*

Proof 1) It is trivial that given the solution of a δ-vertex cover problem, we can quickly ($O(n)$) verify the number of covered vertices is at most k, and for each decomposed connected subgraph (uncovered parts), verify its size is at most δ (e.g. via depth first traversal).

2) For the reduction, we take an instance of a vertex cover and reduce it to a δ-vertex cover instance.

First, assume that we have a vertex cover problem with the graph G. For each vertex in the G, we extend a chain with size $\delta - 1$. Hence, we shall prove that solving the vertex cover problem is same as solving the δ-vertex cover problem. See Figure 5(b).

It is easy to see a vertex cover solution is a correct δ-vertex cover solution, since each covered vertex's has a size $\delta - 1$ chain, and uncovered vertices with its extended chain has size δ and bounded by covering vertices. They are all at most δ size.

Then, we need to prove the δ-vertex cover solution covers the same vertices as the vertex cover solution. For each added chain with original vertex, at most one vertex is needed to be covered, since each chain with original vertex has size δ. Besides, it is not effective to select a vertex in the chain because it cannot fully cover all its neighbor vertices chains. For example, See Figure 5(b): if vertex 1 is selected as covered, it cannot cover vertex 2 and 3, because there are 4 distances to reach. Therefore, we only need cover vertices at original graph G so that it would give same solution as the vertex cover.

5. Experiments

We tested the optimized ordering method on the A* search algorithm (Chen et al., 2018) by tree decomposition. We took the first solution found by the depth-first search as our optimized ordering. We did not compare to the dynamic programming algorithms because they were shown less efficient than A* with naive ordering. We provided Markov networks (triangulated already for consistency) with clique potentials as input files. 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 Results on benchmark models

We first tested the algorithms on several moderate-size benchmark models that are either created from classic Bayesian networks, including Child (Spiegelhalter and Cowell, 1992), Alarm (Beinlich et al., 1989), Barley and Hepar2 (Onisko, 2003), or learned from processed UCI datasets (Lichman, 2013). We first learn a Bayesian network and convert it into an undirected graphical model via the standard moralization procedure (Lauritzen and Spiegelhalter, 1988). Finally, the undirected graphical model is converted into a tree decomposition using a greedy heuristic called min-fill.

In these experiments, we set the $\delta = 3$ and $M = 4$ by default. We chose $\delta = 3$ as it reaches a balance between the diversity and the probability of modes, both of which are necessary for high quality solutions. But in cases A* cannot complete successfully, we adjusted δ down and report the feasible value. Table 6 shows the performance of the competing algorithms on the benchmark models. We list several important properties of the benchmark models that we think affect the running time: the number of variables, max label size, max cluster size, and number of δ-subgraphs. We also list the running time (seconds) and complexities (log scale) of M-Modes A* search before and after optimizing search ordering as well as the running time of the M-Best baseline. The time needed for optimizing the search ordering is included in the total time. We used a depth first search to get the default ordering.

The results show that the theoretical complexity provides a good indication for the amount of running time needed for solving the M-Modes problem. On a same model, the larger the complexity, the more time it takes to solve M-Modes. The results also clearly show that the optimized search orderings did help A* achieve much better efficiency; the improvement ranged from several times

Name	N	δ	Lmax	Clmax	Sub	Time-Df	Cmpl-Df	Time-Opt	Cmpl-Opt
Child	20	3	6	4	92	1.57	26.32	0.19	14.66
Flag	29	3	3	7	133	87.69	41.20	6.37	22.19
Alarm	37	3	4	5	216	0.20	50.94	0.08	22.09
Spectf	45	3	2	9	190	380.15	42.00	16.24	31.00
Barley	48	1	3	8	48	OT	74.49	321.37	45.96
Hepar2	70	1	4	7	70	OT	80.85	3.93	64.51

Name	N	deg	δ	M	Sub	Time-Df	Cmpl-Df	Time-Opt	Cmpl-Opt
Semeion	256	5	6	10	1346	18.37	250.00	3.85	60.00
			6	20	1346	80.71	250.00	10.46	60.00
			7	10	2370	OT	249.00	73.95	67.00

Figure 6: Running time (sec) of M-Modes search on (Top) benchmark models and (Bottom) a real dataset. The first column is the **Name** of models; **N** is the number of variables; δ is the subgraph size; **Lmax** is the max label size of the model; **Clmax** is max cluster size; **deg** is the max tree degree size; And **Sub** is the number of δ-subgraphs. **Time-Df** and **Cmpl-Df** are the searching time and search space complexity for A* with default ordering. **Time-Opt** and **Cmpl-Opt** are the searching time and search space complexity for A* with optimized ordering. OT means time out after half an hour.

to orders of magnitude faster. For the largest datasets (Barley and Hepar2), A* using the default ordering were not able to solve their M-Modes problem within half an hour, but the optimized search ordering enable the algorithm to solve them quickly, even in seconds for Hepar2. However, note that we were only able to solve problems when $\delta = 1$ because M-Modes is challenging to solve.

Semeion is a large UCI dataset with 256 variables, and is used to test the scalability of our methods. We restrict the model to be a tree learned by the Chow-Liu algorithm (Chow and Liu, 1968). The tree had a maximum degree of 5. We varied δ and M in the experiments. The results show that the optimized ordering again showed significant improvement over the default ordering. As δ and M increase, the improvement seems to widen. M-Best timed out on all settings of this dataset.

5.2 Results on random models

To get a more systematic understanding on how the theoretical complexity is affected by different parameters, we also tested the orderings on randomly generated tree decompositions. 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. The default parameter setting for generating tree decompositions is as follows: number of clusters is 6; cluster size is 6, δ is 3, and label size is 2–3. We fix size of separators to be 2 in all experiments. Each time we vary one parameter with others fixed; we generated 100 different junction trees. The results are shown in Figure 7.

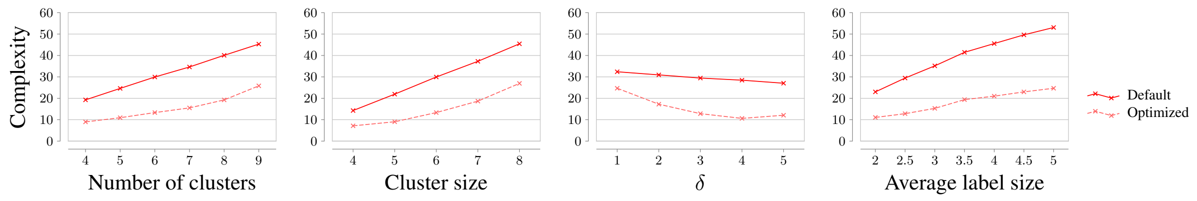


Figure 7: An illustration of how theoretical complexity (log scale) of default ordering and optimized ordering depends on number of clusters, cluster size, δ and average label size.

The results show that in all settings, optimized ordering always reduced the complexity exponentially, with slightly different trends for different parameters. The ordering is getting more effective with increase cluster size, number of clusters and average label size. The effect of δ is more interesting. The increase in δ initially led to reduction in theoretical complexity, but further increase led to rebound in the complexity. The explanation is as follows. When the δ is small, larger δ results in larger δ-subgraphs with more interiors and fewer new boundary variables. However, as δ becomes even larger, close to the cluster size, most δ-subgraphs span multiple clusters, which will include more new boundary variables more quickly.

6. Concluding Remarks

Based on the observation that different search orderings for solving M-Modes have huge impact on the size of search spaces, we 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. We also proposed pruning criteria for speeding up the ordering search.

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, a proper ordering 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.

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.

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 *IJCAI*, pages 3584–3590, 2016.

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

C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *Information Theory, IEEE Transactions on*, 14(3):462–467, 1968.

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 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.

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

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.

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

M. Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`.

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

A. Onisko. Probabilistic causal models in medicine: Application to diagnosis of liver disorders. In *Ph. D. dissertation, Inst. Biocybern. Biomed. Eng., Polish Academy Sci., Warsaw, Poland*, 2003.

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

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