Fast Postprocessing for Difficult Discrete Energy Minimization Problems

Ijaz Akhter
KeepTruckin, Inc
ijaz.akhter@keeptruckin.com

Loong Fah Cheong
National University of Singapore
eleclf@nus.edu.sg

Richard Hartley
Australian National University
Richard.Hartley@anu.edu.au

Abstract

Despite the rapid progress in discrete energy minimization, certain problems involving high connectivity and a high number of labels are considered very hard but are still very relevant in computer vision. We propose a post-processing technique to improve the sub-optimal results of the existing methods on such problems. Our core contribution is a mapping between the binary min-cut problem and finding the shortest path in a directed acyclic graph. Using this mapping, we present an algorithm to find an approximate solution for the min-cut problem. We also extend the same idea for multi-label factor-graphs in the form of an iterative move-making algorithm. The proposed algorithm is extremely fast, yet outperforms the existing techniques in terms of accuracy as well as the computational time. We demonstrate competitive or better results on problems where already high-quality work is done.

1. Introduction

Discrete energy minimization is an extensively studied area in Computer Vision and related fields. In a recent comparative study, a few techniques have been even shown to give globally optimal or near-optimal solutions on several openGM benchmark datasets [9]. However, the problems involving a high number of labels and high-connectivity still remain quite difficult to solve. In addition, in problems where there were no unary terms, such as correlation-clustering, only suboptimal results were obtained. The ability to find good solutions to these more challenging cases is important for better modeling of many computer vision problems, e.g., high connectivity is required to model occlusion in stereo matching [30, 19] and in several other problems including [14, 15, 24, 34]. To improve the accuracy of the solutions for such difficult problems, we propose a fast post-processing algorithm, called ordercut. Ordercut is an iterative move-making algorithm and given the solution of a method as an initialization, it is guaranteed to give non-increasing costs throughout its iterations. We first propose an approximate solution for the binary min-cut problem for an undirected graph, $\mathbf{G}$ and then extend it to general second-order energy minimization problems.

Our core contribution is a mapping between the min-cut problem and finding the shortest path in a directed acyclic graph, $\mathbf{D}$. $\mathbf{D}$ and $\mathbf{G}$ have the same set of nodes but different edge weights. Different orderings of the nodes in $\mathbf{D}$ give different (possibly suboptimal) solutions for the min-cut problem. Starting with an initial labeling, we aim for an ordering such that the nodes that are likely to switch their labels, should move towards the root and the rest away from it, in the DAG. We show that if the weight of the shortest path, in $\mathbf{D}$, is negative then switching the labels of the nodes in that path would decrease the cost of the cut by the weight of the path. The key to the speedup of our approach is the efficient ordering of the nodes and the computation of edge weights in DAG. Finally, we extend order-cut by proposing an iterative move-making algorithm for multi-label factor graphs.

Other move-making algorithms can also be used as a post-processing step to improve the labeling accuracy of an inference technique. But they have their limitations, as we show and discuss in the comparative evaluation. Among these methods, the most prominent are max-flow based move-making algorithms [4], which require sub-modular sub-problems and the truncation technique only works when only a small number of factors are non-sub-modular [31]. The other methods are either slow, not very accurate or work only on a subset of problems.

With the proposed algorithm as a postprocessing technique, we give competitive or outperforming results on challenging problems, where a lot of high-quality previous work has already been done. The proposed method improves our ability to deal with the problems involving high connectivity and a high number of labels. The proposed method is sensitive towards its initialization and mainly meant to be used as post-processing. But we show that even without any good initialization, ordercut gives better results than the previous method on several benchmark datasets including notorious Chinese-Characters and modularity clustering datasets from openGM [9].

2. Related Work

Discrete energy minimization problems are in general NP-hard. A few exceptions include e.g. when the underlying graph structure obeys sub-modularity [16] or when the graph structure is a tree [12]. A classical approach to get an approximate solution is relaxing the integrality constraints [29] (Also see [32]). There also exist specialized algorithms for large-scale problems such as proximal [21, 25], subgradient [17, 10] and quasi-Newton [8]. Although they provide certain convergence guarantees, the complexity of such algorithms can still be very high.

To reduce the computational complexity, certain methods decompose the problem into several tractable subproblems and solve them by the means of Lagrange multipliers. Examples of these methods include Sequential Tree-Reweighted message passing TRWS [13], Max-Product Linear Programming MPLP [5], Adaptive Diminishing Smoothing Algorithm ADSAL [27] and min-sum diffusion [33]. For binary problems, QPBO [23] can be used to efficiently solve the problem. An advantage of linear programming relaxation-based methods is that they normally provide certain bounds to the underlying energy minimization problem. However, if the gap between the estimated energy and the upper bound is large, the solution may be suboptimal. A recent comparative study [9] showed several such examples, such as when the number of labels or the connectivity is large or when the unary is not very discriminative. The focus of our paper is to improve the accuracy of such difficult problems through the proposed post-processing.

Similar to ours, there also exist several move making algorithms. Most popular among them are max-flow based methods, namely $\alpha-$expansion that assumes that the smoothness terms are metric, and $\alpha\beta-$swap that allows the smoothness terms to violate the triangular inequality [16]. A limitation of these methods is that they require sub-modularity and are not applicable to general second-order problems. Other examples include Iterated Conditional Modes ICM, Lazy Flipper LF [1]. Both of these methods can be quite slow for large connectivity and a large number of labels. Recently a problem reduction technique namely PCT [7], was proposed to exploit the structure information in the factor-graph, which allows them to solve smaller subproblems. Their technique, however, is only applicable when the underlying problem is binary, or multi-label potts, though multilabel reduction techniques have also been explored [28]. FastPD [18] lies at the intersection of move-making and dual decomposition-based methods. This method also does not support general second-order factor graphs. In contrast to these methods, the proposed technique is applicable to general second-order graphs and yet quite fast and accurate.

In contrast to all these methods, the global branch and bound type methods such as CombiLP [26] and ILP [11] may be quite slow or even intractable for large scale problems like the one we studied in this paper.

3. Approach

We propose an iterative move-making algorithm to solve second order discrete energy minimization problems. We first convert the binary min-cut problem in an undirected graph, into the shortest-path finding problem in DAG as shown in Fig. 1. The edge weights in DAG are dependent on the ordering of nodes. Given initial labeling of the nodes, we move the nodes, that are likely going to change their labels, towards the root and the rest away from it. With the proposed greedy ordering, we are able to efficiently find an approximate solution to the binary min-cut problem. Then we extend the proposed DAG based shortest path solution for energy minimization in multi-label second-order factor graphs.

3.1. Graph-cut to Shortest Path in DAG Mapping

To define the undirected binary cut problem, we consider a graph $\mathbf{G}$ consisting of $\mathbf{V}$ verticies with edge weights given by a symmetric matrix $\mathbf{W}$. Let there exist two additional nodes, source s and sink t, attached to all the nodes $\mathbf{V}$, with weights given by $\mathbf{s}$ and $\mathbf{t}$ respectively. We define the *Graph-Cut* as a partition of $\mathbf{V}$ into two non-overlapping sets $\mathbf{S}$ (source) and $\mathbf{T}$ (sink) such that $\mathbf{S} \cup \mathbf{T} = \mathbf{V}$ and the cost of the cut is defined as

$$C(\mathbf{S}) = \sum_{x \in \mathbf{T}} \mathbf{s}(x) + \sum_{y \in \mathbf{S}} \mathbf{t}(y) + \sum_{x \in \mathbf{T}, y \in \mathbf{S}} \mathbf{W}(x, y). \quad (1)$$

Given an initial cut $(\mathbf{S}, \mathbf{T})$, the proposed greedy algorithm is to iteratively expand $\mathbf{T}$ or $\mathbf{S}$, such that each expansion decreases the cost of the cut. The algorithm stops when no more expansion move is found. With no initial labeling given, we set $\mathbf{S} = \mathbf{V}$ if $\sum \mathbf{t}(i) < \sum \mathbf{s}(i)$, otherwise $\mathbf{T} = \mathbf{V}$. The following Lemma lays the foundation of the proposed expansion move.

Lemma 3.1. *Let $(\mathbf{S}, \mathbf{T})$ denotes a graph-cut with the cost given by Eqn. 1. Let $(\mathbf{S}', \mathbf{T}')$ denotes another cut, such that a node i is moved from $\mathbf{S}$ to $\mathbf{T}$. Then the cost of the new cut is given by*

$$C(\mathbf{S}') = C(\mathbf{S}) + \mathbf{s}(i) - \mathbf{t}(i) + \sum_{y \in \mathbf{S}} \mathbf{W}(i, y) - \sum_{x \in \mathbf{T}} \mathbf{W}(x, i). \quad (2)$$

The proof is given in Appendix. Lemma 3.1 can be used to devise a greedy algorithm to find a set of nodes such that by moving them from $\mathbf{S}$ to $\mathbf{T}$ (or the other way around) the cost of an existing cut reduces. To exploit this observation we convert the graph $\mathbf{G}$ into a DAG, $\mathbf{D}$ with the following construction.

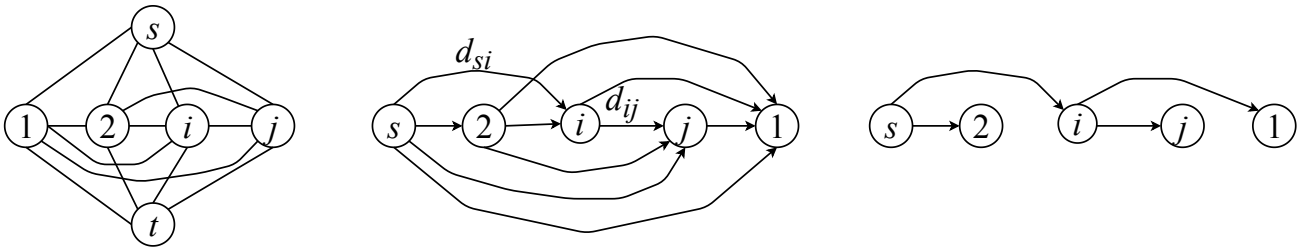

Figure 1. A visualization of the mapping between the undirected graph (left), in which we need to find the min-cut, and the proposed DAG (middle). We map the min-cut problem to the shortest pathfinding problem in DAG, with a sample shortest path tree shown on the right. The DAG contains the same number of nodes as the original graph, with the first node as the source and no sink node. The edge weights in DAG are estimated using Eqns. 3 and 4. These weights are dependent on the ordering of nodes and solving the min-cut problem requires finding an order that gives the minimum shortest path among all possible orderings. With the proposed greedy ordering, given in Eqn. 6, we are able to get an approximate solution for the min-cut problem.

The nodes in $\mathbf{D}$ would be s and $\mathbf{S}$, where s is the source node and $s \notin \mathbf{V}$. The ordered set $\mathbf{P}_{si}$ denotes the shortest path between nodes s and i (excluding s, but including i) with weight $d(\mathbf{P}_{si})$. The first node in $\mathbf{D}$ would be s and connected to all other nodes $i \in \mathbf{S}$ with weights,

$$d_{si} = \mathbf{s}(i) - \mathbf{t}(i) + \sum_{x \in \mathbf{N}(i) \cap \mathbf{S}} \mathbf{W}(i,x) - \sum_{x \in \mathbf{N}(i) \cap \mathbf{T}} \mathbf{W}(i,x), \tag{3}$$

where $\mathbf{N}(i)$ denotes the neighbours of i in $\mathbf{G}$, excluding s. The edge weights between i and $j \in \mathbf{N}(i)$ would be

$$d_{ij} = d_{sj} - 2 \sum_{x \in \mathbf{P}_{si} \cap \mathbf{N}(j)} \mathbf{W}(x,j), \tag{4}$$

where $\mathbf{P}_{si} \cap \mathbf{N}(j)$ is a regular set-intersection by treating the shortest path $\mathbf{P}_{si}$ as an unordered set. As an example, $\mathbf{P}_{si}$ in Fig. 1 just contains node i and not node 2, therefore, here $d_{ij} = d_{sj} - 2 * \mathbf{W}(i,j)$. The dependence of $\mathbf{P}_{si}$ to the estimation of the following edge weights d_{ij} in Eqn. 4 is important to understand and is the reason that different orderings give different shortest paths. Without determining this order, Eqns. 3 and 4 alone are not sufficient to construct DAG. Depending upon the selected order of nodes in $\mathbf{D}$, nodes i and j, can also be disconnected in $\mathbf{G}$. In this case, we find d_{ij} by considering $\mathbf{W}(i,j) = 0$. The following theorem gives the relation between the shortest path and the corresponding cut.

Theorem 3.2. *Given an initial cut* $(\mathbf{S}_0, \mathbf{T}_0)$*, the weight of the path* $\mathbf{P}_{si}$ *and the cost of the cut,* $(\mathbf{S}, \mathbf{T})$*, where* $\mathbf{S} = \mathbf{S_0} - \mathbf{P}_{si}$ *and* $\mathbf{T} = \mathbf{T_0} + \mathbf{P}_{si}$*, are related by*

$$C\left(\mathbf{S}_0 - \mathbf{P}_{si}\right) = C\left(\mathbf{S}_0\right) + d\left(\mathbf{P}_{si}\right), \tag{5}$$

where $\mathbf{S}_0 - \mathbf{P}_{si}$ *is a regular set-difference by treating* $\mathbf{P}_{si}$ *as an unordered set.*

The proof of the theorem is given in Appendix. Once the DAG is built and the shortest path from s to every other node is found, we find the node i_m with the minimum distance $d\left(\mathbf{P}_{si_m}\right)$. If $d\left(\mathbf{P}_{si_m}\right)$ is negative, moving all the nodes in $\mathbf{P}_{si_m}$ from $\mathbf{S}_0$ to $\mathbf{T}_0$ would reduce the cost of the cut by $d\left(\mathbf{P}_{si_m}\right)$. Since the estimation of $\mathbf{P}_{si_m}$ is dependent on the ordering of nodes in $\mathbf{D}$ and different orderings would give different costs, the key challenge is to find an ordering that minimizes the cost of the cut.

Without explicitly building a DAG, we keep the shortest paths of every node in the form of a tree. We initialize the shortest path tree, $\hat{\mathbf{T}}$ with the root node, s. We then add the second node in $\hat{\mathbf{T}}$ with the minimum d_{si}. We denote $\hat{i}$ as the last node added in the tree and $\mathbf{N}(\hat{\mathbf{T}})$ as the nodes adjacent to $\hat{\mathbf{T}}$. We incrementally grow $\hat{\mathbf{T}}$ by adding a node $\hat{j}$ using the following equation,

$$\hat{j} = \underset{j \in \mathbf{N}(\hat{\mathbf{T}})}{\operatorname{argmin}} \left(d(\mathbf{P}_{s\hat{i}}) + d_{\hat{i}j} \right) = \underset{j \in \mathbf{N}(\hat{\mathbf{T}})}{\operatorname{argmin}} d_{\hat{i}j}, \tag{6}$$

where we use Equation 4 to find $d_{\hat{i}j}$. Since negative $d(\mathbf{P}_{s\hat{j}})$ corresponds to the nodes that would reduce the cut-cost by an amount equal to $d(\mathbf{P}_{s\hat{j}})$. The motivation behind Eqn. 6 is to move the nodes that are likely going to switch their labels, towards the source. The shortest path $\mathbf{P}_{s\hat{j}}$ is found by finding the parent node $\hat{k}$ of $\hat{j}$ using the following equation and then attach the both in the current shortest path tree,

$$\hat{k} = \underset{k \in s \cup \hat{i} \cup \hat{\mathbf{T}} \cap \mathbf{N}(\hat{j})}{\operatorname{argmin}} \left(d(\mathbf{P}_{sk}) + d_{k\hat{j}} \right), \tag{7}$$

where the intersection $\hat{\mathbf{T}} \cap \mathbf{N}(\hat{j})$ is taken w.r.t the nodes and the union with s is to handle the case when $d(\mathbf{P}_{s\hat{j}}) = d_{s\hat{j}}$ and the union with $\hat{i}$ is to handle when $\hat{i} \notin \mathbf{N}(\hat{j})$. The above construction only keeps the shortest path from s to every other node and the full DAG is not constructed. Once the tree is fully grown, we find the node i_m with the minimum weight $d(\mathbf{P}_{si_m})$. If that weight is negative, we move all the

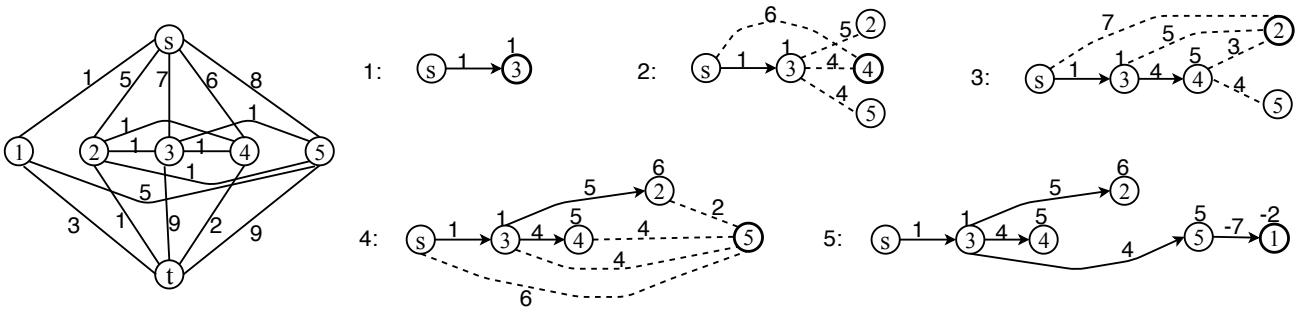

Figure 2. An example to demonstrate the construction of the shortest paths tree given a graph shown on the left. The tree is built in five steps using Eqns. 6 and 7. The detail construction is given in the text.

nodes in $\mathbf{P}_{si_m}$, from the source to the sink. The difference in the cut-cost due to the move would be $d(\mathbf{P}_{si_m})$.

An example of the tree construction and the corresponding estimation of edge weights are given in the example shown in Fig. 2. The graph consists of nodes $\mathbf{V} = \{1, 2, 3, 4, 5\}$ with unary and pairwise terms shown in the graph. For tree construction, we assume an initial cut with $\mathbf{S}_0 = \mathbf{V}$ and $\mathbf{T}_0 = \phi$. Using Eqn. 3, edge weights d_{s1} to d_{s5} are computed as $3, 7, 1, 6, 6$ respectively. After setting $\hat{\mathbf{T}} = s$, the tree is constructed in the following five steps.

1) We attach node 3 to s because it has the minimum d_{si} (See Fig. 2:1). This makes $d(\mathbf{P}_{s3}) = 1$.

2) The next node is selected using Eqn. 6. We find d_{3j}, using Eqn. 4, for $j \in \mathbf{N}(\hat{\mathbf{T}})$, where $\hat{\mathbf{T}} = \{s, 3\}$ and $\mathbf{N}(\hat{\mathbf{T}}) = \{2, 4, 5\}$. This makes $d_{32} = d_{s2} - 2\mathbf{W}(3, 2) = 5$, $d_{34} = d_{s4} - 2\mathbf{W}(3, 4) = 4$ and $d_{35} = d_{s5} - 2\mathbf{W}(3, 5) = 4$. Since d_{34} and d_{35} are the minimums, we arbitrarily select node 4 (shown bold in Fig. 2:2). We find the parent of node 4, using Eqn. 7 as node 3 ($\hat{\mathbf{T}} \cap \mathbf{N}(4) = 3$). Two possible paths from s to node 4 are also shown with dotted line in Fig. 2:2.

3) $\hat{\mathbf{T}} = \{s, 3, 4\}$ and $\mathbf{N}(\hat{\mathbf{T}}) = \{2, 5\}$. Among these two nodes, 2 is selected (shown bold in Fig. 2:3) using Eqn. 6, because $d_{42} = d_{s2} - 2\mathbf{W}(3, 2) - 2\mathbf{W}(4, 2) = 3$ and $d_{45} = d_{s5} - 2\mathbf{W}(3, 5) = 4$ and d_{42} is smaller. Considering all the nodes in $s \cup 4 \cup \hat{\mathbf{T}} \cap \mathbf{N}(2) = \{s, 3, 4\}$ in eqn. 7, the path $(s, 3, 2)$ gives the smallest distance, so we attach node 2 to 3.

4) $\hat{\mathbf{T}} = \{s, 3, 4, 2\}$ and $\mathbf{N}(\hat{\mathbf{T}}) = 5$. $d_{25} = d_{s5} - 2\mathbf{W}(3, 5) - 2\mathbf{W}(2, 5) = 2$. Considering all the nodes in $s \cup 2 \cup \hat{\mathbf{T}} \cap \mathbf{N}(2) = \{s, 3, 4, 2\}$ in eqn. 7, the path $(s, 3, 5)$ gives the smallest distance, so we attach node 5 to 3.

5) $d_{51} = d_{s1} - 2\mathbf{W}(5, 1) = -7$ and the path $(s, 3, 5)$ gives the smallest distance, so we attach node 1 to 5. Among all the nodes, $d(\mathbf{P}_{s1})$ is the minimum and negative ($d(\mathbf{P}_{s1}) = -2$). We switch the labels of the nodes present in the shortest path, $\mathbf{P}_{s1} = (3, 5, 1)$. This gives us $\mathbf{S} = \{2, 4\}$ and $\mathbf{T} = \{1, 3, 5\}$.

The construction of the shortest path tree in the above example is quite inefficient and can be significantly speedup. In particular, the edge weights used in Eqn. 7 have already been computed when the earlier nodes were added in the tree. Eqns. 6 and 7 can be solved efficiently with the help of Priority Queue based indexing, as we show next. Finally, the intersection $\mathbf{P}_{si} \cap \mathbf{N}(j)$ can also be found quite efficiently, as we show in Sec. 3.3

3.2. Expansion Algorithm

To efficiently find node $\hat{j}$ using Eqn. 6, we store the neighbors of nodes currently added in the tree in the form of an indexed priority queue. The queue sorts them based on $d_{\hat{i}j}$ and they are indexed by $\hat{i}$. Therefore, if multiple edges with the same ID are pushed, only the last one is retained. Index priority queue gives the minimum weight edge in constant time, whereas insert, delete and update operations are done in logarithmic time. Algorithm 1 outlines the shortest path tree construction algorithm. The efficient estimation of edge weights and an early termination of the algorithm are discussed in the next section. Once tree is constructed, we find $\mathbf{P}_{si_m}$ if $d(\mathbf{P}_{si_m}) < 0$, and relabel the corresponding nodes. We rerun the algorithm with the updated labeling to see if the cost can further be reduced, otherwise, we terminate.

The input to the Algorithm 1 includes edge weights $\mathbf{W}$ in $\mathbf{G}$, current labels $\mathbf{L}$, indexed priority queue $\mathbf{Q}$, current path weights for all the nodes $\mathbf{d}$ (initially these distances are set to d_{si}), a boolean array $\mathbf{vstd}$ with value 1 at its index i if the node i was visited before and have already changed its label, the $\mathbf{tree}$ data structure, initialized with only the root node. Given these as input, we set $\hat{i}$ equal to the root of the tree. We pop the minimum weight edge on line 4, where $\hat{j}$ is the ending node of the edge and $\hat{v}$ contains the weight and the starting node as $\hat{v} = \{w, i\}$. If node $\hat{j}$ is not in the tree yet (lines 5-7), then we check, if we should attach $\hat{j}$ to the last visited node, $\hat{i}$ or to the starting node of the edge, $\hat{v}.i$, in the $\mathbf{tree}$ to get smaller $d(\mathbf{P}_{s\hat{j}})$ (lines 8-13). This does not exactly correspond to Eqn. 7, but it makes the algorithm efficient at the cost of a little sub-optimality. We update $\mathbf{d}$

Algorithm 1: Shortest path tree construction

1 function construct-tree $(\mathbf{W}, \mathbf{L}, \mathbf{Q}, \mathbf{d}, \mathbf{vstd}, \mathbf{tree})$;
// $\mathbf{Q}$ contains a list of tree-edges, sorted by their weights and indexed by the edge's ending node, where Tree-Edge = $\{w, i\}$, and w :weight & i : starting node of the edge.
2 $\hat{i} = \mathbf{tree}.root$;
3 **while** !$\mathbf{Q}.isempty()$ **do**
4 $\quad \left[\hat{j}, \hat{v}\right] = \mathbf{Q}.popmin()$;
5 $\quad$ **if** $\mathbf{vstd}[\hat{j}]$ **then**
6 $\quad\quad$ **continue**;
7 $\quad$ **end**
8 $\quad \hat{d} = \mathbf{d}[\hat{i}] + \hat{v}.w$;
9 $\quad d = \mathbf{d}[\hat{v}.i] + \hat{v}.w$;
10 $\quad$ **if** $d < \hat{d}$ **then**
11 $\quad\quad \hat{i} = \hat{v}.i; \qquad \hat{d} = d$;
12 $\quad$ **end**
13 $\quad \mathbf{tree}.addedge(\hat{i}, \hat{j}, \hat{d})$;
14 $\quad \mathbf{d}[\hat{j}] = \hat{d}$;
15 $\quad \mathbf{vstd}[\hat{j}] = 1$;
16 $\quad$ **for** $k \in \mathbf{N}(\hat{j})$ & $\mathbf{vstd}[k] = 0$ **do**
17 $\quad\quad$ find $d_{\hat{j}k}$ using Eqn. 4;
18 $\quad\quad \mathbf{Q}.push(k, \{d_{\hat{j}k}, \hat{j}\})$;
19 $\quad$ **end**
20 $\quad \hat{i} = \hat{j}$;
21 **end**

Algorithm 2: Expansion move for $\mathbf{T}$

1 function OC-Expand-Sink $(\mathbf{s}, \mathbf{t}, \mathbf{W}, \mathbf{L})$;
Input : // source, sink & pairwise weights, & initial labeling
Output: $\mathbf{L}$ // $\mathbf{L}[i] = 0$ if $i \in \mathbf{S}$, & 1 if $i \in \mathbf{T}$
2 $n = length(\mathbf{s}), \qquad \alpha = 2$
3 Initialize an Indexed Proirity Queue for Tree-Edges, $\mathbf{Q}$ of capacity n.
4 $\forall i \in \{1, \ldots, n\}, \mathbf{vstd}[i] = (\mathbf{L}[i] == 0)\ ?\ 0 : 1$;
5 $\forall i \in \{1, \ldots, n\}, \mathbf{d}[i] = (\mathbf{L}[i] == 0) ? d_{si} : \infty$;
// find d_{si} using Eq. 3
6 **for** *i from the first α percentiles of* $\mathbf{D}$ **do**
7 $\quad \mathbf{Q}.push(i, \{d_{si}, s\})$;
8 **end**
9 $\hat{i} = s$; // node to start tree growing
10 **while** *true* **do**
11 $\quad \mathbf{tree}.root = \hat{i}$ // create a tree with root $\hat{i}$
12 $\quad$ construct-tree($\mathbf{W}, \mathbf{L}, \mathbf{Q}, \mathbf{d}, \mathbf{vstd}, \mathbf{tree}$);
13 $\quad$ find the node i_m with the min-weight d_m in $\mathbf{tree}$;
14 $\quad$ **if** $d_m < 0$ **then**
15 $\quad\quad$ find the path $\mathbf{P}_{si_m}$ connecting i_m and s;
16 $\quad\quad$ **for** $j \in \{1, \ldots, n\}$ **do**
17 $\quad\quad\quad$ **if** $j \in \mathbf{P}_{si_m}$ **then**
18 $\quad\quad\quad\quad \mathbf{L}[j] = 1$;
19 $\quad\quad\quad$ **end**
20 $\quad\quad\quad$ **else**
21 $\quad\quad\quad\quad \mathbf{d}[i] = d_{si}$(using Eq. 3);
22 $\quad\quad\quad\quad \mathbf{vstd}[j] = 0$;
23 $\quad\quad\quad$ **end**
24 $\quad\quad$ **end**
25 $\quad\quad$ delete $\mathbf{tree}$;
26 $\quad\quad \hat{i} = i_m$;
27 $\quad\quad$ push the first α percentiles of $\mathbf{d}$ in $\mathbf{Q}$;
28 $\quad$ **end**
29 $\quad$ **else**
30 $\quad\quad$ **break**;
31 $\quad$ **end**
32 **end**

with the estimated path-weight (line 14). Then we update the weights, $d_{\hat{j}k}$ for $k \in \mathbf{N}(\hat{j})$ (lines 16-19). If all the nodes have been added to the tree, we terminate the while loop (line 3).

Shortest path tree construction is used as a sub-routine in the proposed expansion move algorithm 1. Given an initial labeling, we are expanding the set $\mathbf{T}$ (nodes with label equal to zero). Swapping $\mathbf{s}$ with $\mathbf{t}$ and switching the corresponding labels would make it work to expand $\mathbf{S}$. Indexed Priority Queue, $\mathbf{Q}$ for edges and $\mathbf{vstd}$ array, to keep track of the nodes that have been added to the DAG, are initialized on line 3-4. The vector $\mathbf{d}$ is initialized with edge weight d_{si} (line 5). We push the first α percentiles of edges based on their weights (d_{si}) in $\mathbf{Q}$, where $\alpha = 2$ (lines 6-8). The while-loop (lines 10-31) runs until no shortest path with a negative weight is found. We initialize a tree with the root $\hat{i}$ (line 11). Then call the **construct-tree** function, with the current set of variables, to get the shortest paths from the source to every other node (line 12). After this, we find the node i_m with the minimum weight (line 13). If d_m is negative, we find the path $\mathbf{P}_{si_m}$ (lines 14-15) otherwise we terminate the loop (line 30). We change the labels of nodes in $\mathbf{P}_{si_m}$ from 0 to 1 (line 18). For the rest of the nodes, we reestimate d_{si} according to the updated labeling and reset them as unvisited (lines 21-22). We delete the current tree and reset its root for the next iteration (lines 25-26). We also push the first α percentiles of edges based on their weights (d_{si}) in $\mathbf{Q}$ (line 27). Upon termination, the final labels are stored in $\mathbf{L}$.

3.3. Optimization Details

In the previous section, the main problem in efficiently finding the solution is, how to find the edge weights, $d_{\hat{j}k}$ on line 17 of the Algorithm 1? Since we store the shortest paths in the form a tree, finding $\mathbf{P}_{s\hat{j}} \cap \mathbf{N}(k)$ in Eqn. 4, requires traversing the whole path from s to $\hat{j}$ and then taking its intersection with $\mathbf{N}(k)$. To do this efficiently, we exploit the fact that most of the time, a major part of the tree is a linear link-list. We keep a record of that list in the form of a binary occupancy array, of size n, with value 1, if the corresponding node lies on the current path $\mathbf{P}_{s\hat{j}}$ and 0 otherwise. $\mathbf{P}_{s\hat{j}} \cap \mathbf{N}(k)$ can now easily be found by checking if for a node in $\mathbf{N}(k)$ the corresponding entry in the occupancy array is 1 or not. On line 10 of the Algorithm 1, if $d < \hat{d}$ then the occupancy array can be used to find $\mathbf{P}_{s\hat{j}} \cap \mathbf{N}(k)$, otherwise, node $\hat{j}$ would create a parallel edge in the tree, and we need to update the occupancy array corresponding to the path $\mathbf{P}_{s\hat{j}}$. This, however, does not happen very often, making the proposed algorithm quite fast.

Under the assumption that the most part of the tree is a linear link-list, we can find the weights $d_{\hat{j}k}$ even more efficiently. We need to initialize an array $\mathbf{b}$, of size n with zeros. With each addition of node $\hat{j}$, we can efficiently find $d_{\hat{j}k}$ as following,

$$\mathbf{b}[k] = \mathbf{b}[k] - 2\mathbf{W}(\hat{j}, k), \tag{8}$$

$$d_{\hat{j}k} = d_{s\hat{j}} + \mathbf{b}[k], \text{ where } k \in \mathbf{N}(\hat{j}). \tag{9}$$

In contrast to Eqn. 4, which requires running a loop over the set of neighbours, Eqn. 9 contains just one subtraction and one multiplication, and is more efficient. Whenever, we add a parallel edge, we have to update the occupancy array by considering the linear list between the source and the parallel edge. We put the new nodes corresponding to the parallel edge in a $to\text{-}add$ list, and the nodes we have to delete in a $to\text{-}delete$ list. We update $\mathbf{b}$ as $\mathbf{b}[k] = \mathbf{b}[k] - 2\mathbf{W}(\hat{j}, k)$ for $k \in to\text{-}add$, and $\mathbf{b}[k] = \mathbf{b}[k] + 2\mathbf{W}(\hat{j}, k)$ for $k \in to\text{-}delete$ nodes.

Lastly, early termination of the proposed expansion algorithm can also save the computation time. To do this, we find the current minimum weight as $\hat{d}_m$ and the corresponding node, $\hat{j}_m$ for each addition of a node in the tree. If for the current node, $\hat{d} - \hat{d}_m$ is greater than a threshold and a certain number of nodes has been added in the tree since the last $\hat{j}_m$, we terminate the tree construction.

3.4. Extension for Multi-label Problems

The energy minimization for the second order multi-label problems is defined as,

$$\hat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} \sum_i \mathbf{U}_{x_i}(i) + \sum_{i,j \in \mathbf{N}(i)} \mathbf{W}_{x_i,x_j}(i,j), \tag{10}$$

where $\mathbf{x}$ denotes the set of labels for all the nodes, with each x_i assuming L possible labels, and $\mathbf{U}$ and $\mathbf{W}$ are the unary and pairwise terms. Even though, they look different, but equations 1 and 10 are quite similar. This can be seen once we evaluate both the energy terms given a labeling of nodes. Hence, by generalizing equations 3 and 9, the proposed expansion algorithm can be generalized to the multi-label problem. Assuming initially $x_i = l'$, the edge weights d^l_{si} can be found by noticing the difference in the energy term, if x_i is changed to l as follows,

$$d^l_{si} = \mathbf{U}_l(i) - \mathbf{U}_{l'}(i) + \sum_{j \in \mathbf{N}(i)} \left(\mathbf{W}_{lx_j}(i,j) - \mathbf{W}_{l'x_j}(i,j) \right), \tag{11}$$

Similarly, if $x_j = l''$, for a node $j \in \mathbf{N}(i)$, then d^l_{ij} can also be found, by expanding the energy term in Eqn. 10 and noticing the difference from the starting energy as follows,

$$\mathbf{b}[j] = \mathbf{b}[j] + \mathbf{W}_{ll}(i,j) - \mathbf{W}_{l'l}(i,j) - \mathbf{W}_{ll''}(i,j) + \mathbf{W}_{l'l''}(i,j), \tag{12}$$

$$d^l_{ij} = d^l_{sj} + \mathbf{b}[j], \text{ where } j \in \mathbf{N}(i). \tag{13}$$

Hence the proposed expansion move can be extended for multi-label problems. Specifically, the proposed full algorithm is iterative, where within each iteration, we run our expansion algorithm for each label. If no initial labeling is given, we set the labels of all the nodes to,

$$\hat{l} = \underset{l}{\operatorname{argmin}} \sum_i \mathbf{U}_l(i). \tag{14}$$

4. Evaluation

To evaluate the proposed approach, we use the datasets from OpenGM [9] benchmark that had a large gap between the reported energy value and the corresponding lower bound for the tested polyhedral methods. The large gap indicates that the estimated labels may be suboptimal and there is room for improvement. These datasets have either a large number of labels or large connectivity or less informative unaries in comparison with the smoothness terms. We compare the performance of Order-Cut (OC) against previous move-making algorithms, namely iterative conditional modes (ICM) [2], lazy flipper (LF) [1], local submodular approximations (LSA-TR) [6], and Kernighan-Lin algorithm (KLA) [20] and α-expansion[4]. We use OpenGM library for ICM, LF, and KLA and MRF library for α-expansion from [31]. For the rest of the methods, we use their publicly available source codes.

In Tab. 1, we evaluate OC, ICM and LF as post-processing algorithms, initialized on TRWS [13], BPS[22], and $\alpha-$ expansion on mrf-stereo dataset. We report the average final energy value for each algorithm. This dataset has grid4 connectivity with 16-60 labels and roughly 150,000

	TRWS		BPS		$\alpha-$Exp	
	energy	time(sec)	energy	time(sec)	energy	time(sec)
	1659616.0	4.9	1793082.0	4.4	1606607.0	8.7
+OC	**1617548.3**	**3.4**	**1744174.0**	**3.7**	**1606207.7**	**0.5**
+ICM	1644089.7	9.2	1779120.0	9.0	1612515.0	8.8
+LF2	1630915.3	336.4	1768286.7	339.2	1610021.3	313.5

Table 1. *mrf-stereo: This dataset has 3 instances with 16-60 labels and roughly 150,000 nodes. The large label space makes the problem difficult. The first row gives the energy and computation time of TRWS, BPS, and $\alpha-$Exp. The following rows are the results due to the post-processing by OC, ICM, and LF2. OC performs better than ICM and LF2 while still is considerably faster.*

	TRWS		BPS	
	energy	time	energy	time
	19088916	6.6	19090723	7.3
+OC	19088403	**2.7**	19090685	**2.3**
+ICM	19088445	17.7	19090691	17.8
+LF2	**19088174**	100	**19090678**	101

Table 2. *brain-5mm (4 instances): The required rounding on the continuous solution estimated using TRWS and BPS, in not trivial in this dataset. This issue can be solved using the proposed method very efficiently.*

	TRWS		BPS	
	energy	time(sec)	energy	time(sec)
	-5228.2	8.3	-5768.9	9.7
+OC	-5351.4	1.8	-5808.5	1.8

Table 3. *protein-folding (21 instances): This dataset had 81-503 labels and high connectivity, making the problem very hard. But still, OC significantly improves the results in less than 2 seconds, on average.*

	OC	ICM	LF2
energy	**19127454**	19272820	19140692
time(sec)	**26.5**	33.9	118.5

Table 4. *brain-5mm: Comparative Evaluation of OC, ICM, and LF, with no initialization.*

	OC	LF1	KL
energy	**-26003.1**	-25243.8	-25557.1
time(sec)	**2.5**	63.7	10.5

Table 5. *knott-3d-300 (8 instances): This dataset had 3846-5896 nodes and the same number of labels and there are no unary terms, making it hard to infer.*

	OC	ICM	LF2	LSA-TR
energy	**-49543.5**	-49516.0	-49531.1	-49533.0
time(sec)	**0.14**	0.70	22.44	0.18

Table 6. *dtf-chinesechar (100 instances): This is a binary problem but each node has 27 neighbors. OC still outperforms others both in terms of speed as well as accuracy.*

nodes. We use a depth of two for LF. With a large number of labels, higher depth LF becomes intractable. With all three initializations, OC works better than ICM and LF, while is considerably faster. For such problems where the connectivity of nodes is small and the edge-weights are positive and pairwise terms are more powerful than the unary terms, OC as a stand-alone method performs worse than the conventional state of the art methods like $\alpha-$expansion, $\alpha-\beta$ swap, TRWS, and BPS.

In Tab. 2, we report the same comparison on brain-5mm dataset, containing 1413972 nodes with 3D-grid6 connectivity and five labels. TRWS provides tighter bound in this case, but rounding off the solution is non-trivial. LF with depth 2 provides better results but is very slow. OC provides better results than ICM, while still being very fast. The global optimal result reported in [9] on this dataset was 19087612.5 using Integer Multiway Cut [11].

In Tab. 3, we report results on the protein-folding dataset, showing the effectiveness of the proposed method. These results show that OC is more accurate and considerably faster than ICM and LF.

In Tab. 4, we report results on the Brain-5mm dataset, when no initialization was given. Since the edge weights here are positive, α-expansion can also be tested. However, we could not run α-expansion in the MRF-library [31] because it does not support general sparsely connected factor graphs, (except grid-4 and grid 8 graphs in 2D). However, from Tab. 11 in [9], we can see that α-exp-VIEW on this dataset is more accurate (19089080) but quite slow (100 seconds).

In Tabs. 5 and 6, we do the same comparison on knott-3d-300 and dtf-chinesechar. Knott-3d-300 is a correlation clustering[1] dataset with 3846-5896 nodes and the same number of labels and very high connectivity. With no unary terms, polyhedral methods fail to give a tight poly-type relaxation and because of negative pairwise terms, max-flow based methods are not applicable. Dtf-chinesechar is a binary label Chinese characters inpainting dataset with each node having 27 neighbors. In Tab. 5, we use lazy flipper with a depth of 1 because depth equal to 2 was intractable because of large label space and connectivity. OC demonstrates better accuracy, with a considerable speedup on both brain-5mm and knott-3d-300 dataset. In Tab. 6, we also

[1]We did not use the other two correlation clustering dataset from OpenGM, because they seem to be erroneous. With no negative pairwise terms, the trivial solution of a single cluster becomes the optimal solution.

compare our results with LSA-TR with Hamming distance [6]. In their original paper, they report slightly better results with Euclidean distance, but that requires initializing with a 2D array of labels. Whereas in a more general scenario, where nodes may not lie on a grid structure, only a 1D initialization can be provided. The best-reported results on this dataset in [9] were -49550 using MaxCut Branch and Cut method [3] though optimality was not guaranteed. OC gives quite close results while being significantly faster.

Please note that in these tables, we do not compare our results with PCT [7] as a postprocessing tool because it only works with binary problems or multilabel potts, which means it is only applicable on brain-5mm and dtf-chinesechar. Comparing our results with Tab. 11 and 12 in [9], we see that TRWS-PCT results on brain-5mm are slower (22 seconds vs. ours 10 seconds) but more accurate (19087728 vs. ours 19088916), whereas for dtf-chinesechar TRWS-PCT is worse than OC alone (-49497 vs. ours -49543) and is also slower (4.4 seconds vs. ours 0.14 seconds)

All these examples demonstrate the utility of the ordercut as post-processing for second-order discrete energy minimization problems when polyhedral relaxation-based methods fail to give a tight bound on the actual energy function.

5. Conclusion

We propose a fast postprocessing technique to improve the suboptimal results of the existing methods on difficult discrete energy minimization problems. We show that the proposed technique is useful for problems with high connectivity or a high number of labels. High connectivity arises in many computer vision problems e.g. in occlusion handling in stereo. Existing polyhedral techniques also fail to give accurate results when local polytype relaxation is not very tight, e.g. when there are no unary terms. For such scenarios, we provide a fast method to improve labeling accuracy. The proposed method, as a stand-alone method, performs worse on problems where connectivity is small and smoothness terms are more powerful than the unary terms. Extending our approach for higher-order terms is a future direction.

6. Acknowledgement

This work was supported by the Australian Research Center grant ARCDP13 and Singapore PSF grant 1521200082.

7. Appendix

Proof. (Lemma 3.1) Using Equation 1 for the cut $(\mathbf{S}', \mathbf{T}')$,

$$C(\mathbf{S}') = \sum_{x\in\mathbf{T}'} \mathbf{s}(x) + \sum_{y\in\mathbf{S}'} \mathbf{t}(y) + \sum_{x\in\mathbf{T}', y\in\mathbf{S}'} \mathbf{W}(x,y).$$

Substituting $\mathbf{S}' = \mathbf{S} - i$ and $\mathbf{T}' = \mathbf{T} \cup i$,

$$\begin{aligned} C(\mathbf{S}') &= \sum_{x\in\mathbf{T}} \mathbf{s}(x) + \mathbf{s}(i) + \sum_{y\in\mathbf{S}} \mathbf{t}(y) - \mathbf{t}(i) \\ &\quad + \sum_{x\in\mathbf{T}, y\in\mathbf{S}} \mathbf{W}(x,y) + \sum_{y\in\mathbf{S}} \mathbf{W}(i,y) - \sum_{x\in\mathbf{T}} \mathbf{W}(x,i) \\ &= C(\mathbf{S}) + \mathbf{s}(i) - \mathbf{t}(i) + \sum_{y\in\mathbf{S}} \mathbf{W}(i,y) - \sum_{x\in\mathbf{T}} \mathbf{W}(x,i) \end{aligned}$$

□

Proof. (Theorem 3.2) We prove the theorem by induction. To prove the base case, we need to show that if $\mathbf{P}_{si} = i$, then $d(\mathbf{P}_{si}) = d_{si}$. Since $\mathbf{W}(i,j) = \mathbf{W}(j,i)$ and $\mathbf{W}(i,j) = 0$ for $j \notin \mathbf{N}(i)$, using Lemma 3.1, we can write

$$\begin{aligned} C(\mathbf{S}_0 - i) &= C(\mathbf{S}_0) + \mathbf{s}(i) - \mathbf{t}(i) + \sum_{x\in\mathbf{N}(i)\cap\mathbf{S}_0} \mathbf{W}(i,x) \\ &\quad - \sum_{x\in\mathbf{N}(i)\cap\mathbf{T}_0} \mathbf{W}(i,x) = C(\mathbf{S}_0) + d_{si}. \end{aligned}$$

This proves the base case. In the inductive step, we assume that the Equation 5 is true for $i < k$, therefore,

$$C\left(\mathbf{S}_0 - \mathbf{P}_{si}\right) = C\left(\mathbf{S}_0\right) + d\left(\mathbf{P}_{si}\right), \forall i < k \qquad (15)$$

Consider the node k attached to the nodes $i < k$ with weight given using Eqn. 4

$$\begin{aligned} d_{ik} &= d_{sk} - 2 \sum_{x\in\mathbf{P}_{si}\cap\mathbf{N}(k)} \mathbf{W}(x,k) = d_{sk} - 2 \sum_{x\in\mathbf{P}_{si}} \mathbf{W}(x,k), \\ &= \mathbf{s}(k) - \mathbf{t}(k) + \sum_{x\in\mathbf{S}_0} \mathbf{W}(k,x) - \sum_{x\in\mathbf{T}_0} \mathbf{W}(k,x) \\ &\quad - 2 \sum_{x\in\mathbf{P}_{si}} \mathbf{W}(x,k) \\ &= \mathbf{s}(k) - \mathbf{t}(k) + \sum_{y\in\mathbf{S}_0-\mathbf{P}_{si}} \mathbf{W}(k,y) - \sum_{x\in\mathbf{T}_0+\mathbf{P}_{si}} \mathbf{W}(x,k). \end{aligned}$$

According to the Lemma 3.1, if there exists a cut $(\mathbf{S}_0 - \mathbf{P}_{si}, \mathbf{T}_0 + \mathbf{P}_{si})$ and a node k is moved from the source to the sink then the cost of the new cut is given by

$$C\left(\mathbf{S}_0 - \{\mathbf{P}_{si}, k\}\right) = C\left(\mathbf{S}_0 - \mathbf{P}_{si}\right) + d_{ik}.$$

Using Equation 15,

$$C\left(\mathbf{S}_0 - \{\mathbf{P}_{si}, k\}\right) = C\left(\mathbf{S}_0\right) + d\left(\mathbf{P}_{si}\right) + d_{ik}. \qquad (16)$$

The term, $d\left(\mathbf{P}_{si}\right) + d_{ik}$ in the above equation, gives the costs of different paths from s to k for different $i < k$ and the L.H.S is the corresponding cost for the cut $(\mathbf{S}_0 - \{\mathbf{P}_{si}, k\}, \mathbf{T}_0 + \{\mathbf{P}_{si}, k\})$. Among all these paths, the shortest path $\mathbf{P}_{sk}$ can be found by minimising the above equation w.r.t i as follows,

$$C\left(\mathbf{S}_0 - \{\mathbf{P}_{si}, k\}\right) = C\left(\mathbf{S}_0\right) + d\left(\mathbf{P}_{sk}\right). \qquad (17)$$

□

References

[1] B. Andres, J. H. Kappes, T. Beier, U. Köthe, and F. A. Hamprecht. The lazy flipper: Efficient depth-limited exhaustive search in discrete graphical models. In *European Conference on Computer Vision*, pages 154–166. Springer, 2012.

[2] J. Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society: Series B (Methodological)*, 48(3):259–279, 1986.

[3] T. Bonato, M. Jünger, G. Reinelt, and G. Rinaldi. Lifting and separation procedures for the cut polytope. *Mathematical Programming*, 146(1-2):351–378, 2014.

[4] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 1, pages 377–384. IEEE, 1999.

[5] A. Globerson and T. S. Jaakkola. Fixing max-product: Convergent message passing algorithms for map lp-relaxations. *Advances in neural information processing systems*, pages 553–560, 2008.

[6] L. Gorelick, Y. Boykov, O. Veksler, I. B. Ayed, and A. Delong. Local submodularization for binary pairwise energies. *IEEE transactions on pattern analysis and machine intelligence*, 39(10):1985–1999, 2017.

[7] J. Hendrik Kappes, M. Speth, G. Reinelt, and C. Schnorr. Towards efficient and exact map-inference for large scale discrete computer vision problems via combinatorial optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1752–1758, 2013.

[8] H. Kannan, N. Komodakis, and N. Paragios. Newton-type methods for inference in higher-order markov random fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7300–7309, 2017.

[9] J. Kappes, B. Andres, F. Hamprecht, C. Schnorr, S. Nowozin, D. Batra, S. Kim, B. Kausler, J. Lellmann, N. Komodakis, et al. A comparative study of modern inference techniques for discrete energy minimization problems. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1328–1335, 2013.

[10] J. H. Kappes, B. Savchynskyy, and C. Schnörr. A bundle approach to efficient map-inference by lagrangian relaxation. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1688–1695. IEEE, 2012.

[11] J. H. Kappes, M. Speth, G. Reinelt, and C. Schnörr. Higher-order segmentation via multicuts. *Computer Vision and Image Understanding*, 143:104–119, 2016.

[12] D. Koller, N. Friedman, and F. Bach. Probabilistic graphical models: principles and techniques. In *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[13] V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE transactions on pattern analysis and machine intelligence*, 28(10):1568–1583, 2006.

[14] V. Kolmogorov and C. Rother. Comparison of energy minimization algorithms for highly connected graphs. In *European Conference on Computer Vision*, pages 1–15. Springer, 2006.

[15] V. Kolmogorov and R. Zabih. Multi-camera scene reconstruction via graph cuts. In *European conference on computer vision*, pages 82–96. Springer, 2002.

[16] V. Kolmogorov and R. Zabih. What energy functions can be minimizedvia graph cuts? *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (2):147–159, 2004.

[17] N. Komodakis, N. Paragios, and G. Tziritas. Mrf energy minimization and beyond via dual decomposition. *IEEE transactions on pattern analysis and machine intelligence*, 33(3):531–552, 2011.

[18] N. Komodakis and G. Tziritas. Approximate labeling via graph cuts based on linear programming. *IEEE transactions on pattern analysis and machine intelligence*, 29(8):1436–1453, 2007.

[19] M. H. Lin and C. Tomasi. Surfaces with occlusions from layered stereo. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 1, pages I–I. IEEE, 2003.

[20] S. Lin. An efficient heuristic algorithm for the traveling salesman problem. *Operations Research*, 21:498–516, 1973.

[21] O. Meshi and A. Globerson. An alternating direction method for dual map lp relaxation. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 470–483. Springer, 2011.

[22] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

[23] C. Rother, V. Kolmogorov, V. Lempitsky, and M. Szummer. Optimizing binary mrfs via extended roof duality. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007.

[24] C. Rother, S. Kumar, V. Kolmogorov, and A. Blake. Digital tapestry [automatic image synthesis]. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 589–596. IEEE, 2005.

[25] B. Savchynskyy, J. Kappes, S. Schmidt, and C. Schnörr. A study of nesterov's scheme for lagrangian decomposition and map labeling. In *CVPR 2011*, pages 1817–1823. IEEE, 2011.

[26] B. Savchynskyy, J. H. Kappes, P. Swoboda, and C. Schnörr. Global map-optimality by shrinking the combinatorial search area with convex relaxation. In *Advances in Neural Information Processing Systems*, pages 1950–1958, 2013.

[27] B. Savchynskyy, S. Schmidt, J. Kappes, and C. Schnörr. Efficient mrf energy minimization via adaptive diminishing smoothing. *arXiv preprint arXiv:1210.4906*, 2012.

[28] A. Shekhovtsov, P. Swoboda, and B. Savchynskyy. Maximum persistency via iterative relaxed inference with graphical models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 521–529, 2015.

[29] M. I. Shlezinger. Syntactic analysis of two-dimensional visual signals in the presence of noise. In *Cybernetics and systems analysis*, pages 612–628, 1976.

[30] J. Sun, N.-N. Zheng, and H.-Y. Shum. Stereo matching using belief propagation. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (7):787–800, 2003.

[31] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother. A comparative study of energy minimization methods for markov

random fields. In *European conference on computer vision*, pages 16–29. Springer, 2006.

[32] T. Werner. A linear programming approach to max-sum problem: A review. *Research Reports of CMP*, 2005.

[33] T. Werner and D. Pru?sa. The power of lp relaxation for map inference. *Advanced Structured Prediction*, page 19, 2014.

[34] Z. Zabih. Computing visual correspondence with occlusions using graph cuts. In *Eighth IEEE International Conference on Computer Vision*, volume 2, pages 508–515, 2001.