Memory Efficient Max Flow for Multi-label Submodular MRFs

Thalaiyasingam Ajanthan, Richard Hartley
Australian National University & NICTA*
Canberra, Australia

Mathieu Salzmann
CVLAB, EPFL
Lausanne, Switzerland

Abstract

Multi-label submodular Markov Random Fields (MRFs) have been shown to be solvable using max-flow based on an encoding of the labels proposed by Ishikawa, in which each variable X_i is represented by ℓ nodes (where ℓ is the number of labels) arranged in a column. However, this method in general requires $2\,\ell^2$ edges for each pair of neighbouring variables. This makes it inapplicable to realistic problems with many variables and labels, due to excessive memory requirement. In this paper, we introduce a variant of the max-flow algorithm that requires much less storage. Consequently, our algorithm makes it possible to optimally solve multi-label submodular problems involving large numbers of variables and labels on a standard computer.

1. Introduction

Ishikawa [13] introduced a max-flow-based method to globally minimize the energy of multi-label MRFs with convex edge terms. In [22], this method was extended to energy functions satisfying the *multi-label submodularity* condition, analogous to the submodularity condition for MRFs with binary labels. In the general case, however, this method requires $2\,\ell^2$ directed edges for each pair of neighbouring variables. For instance, for a 1000×1000, 4-connected image with 256 labels, it would require approximately $1000{\times}1000{\times}2{\times}256^2{\times}2{\times}4 \approx 1000$ GB of memory to store the edges (assuming 4 bytes per edge). Clearly, this is beyond the storage capacity of most computers.

In this paper, we introduce a variant of the max-flow algorithm that requires storing only two ℓ-dimensional vectors per variable pair instead of the $2\,\ell^2$ edge capacities of the standard max-flow algorithm. In the example discussed above, our algorithm would therefore use only 4 GB of memory for the edges. As a result, our approach lets us optimally solve much larger problems.

More specifically, in contrast to the usual augmenting path algorithm [8], we do not store the residual edge capacities at each iteration. Instead, our algorithm records two ℓ-dimensional flow-related quantities for every pair of neighbouring variables. We show that, at any stage of the algorithm, the residual edge capacities can be computed from these flow-related quantities and the initial edge capacities. This, of course, assumes that the initial capacities can be computed by some memory-efficient routine, which is almost always the case in computer vision.

The optimality of Ishikawa's formalism made it a method of choice as a subroutine in many approximate energy minimization algorithms, such as multi-label moves [26, 27] and IRGC [1]. Since our approach can simply replace the standard max-flow algorithm [5] in Ishikawa-type graphs, it also allows us to minimize the energy of much larger non-submodular MRFs in such approximate techniques. Furthermore, due to the similarity to standard max-flow, our algorithm can easily be extended to handle dynamic MRFs [16] and also be accelerated using the parallel max-flow technique [24].

We demonstrate the effectiveness of our algorithm on the problems of stereo correspondence estimation and image inpainting. Our experimental evaluation shows that our method can solve much larger problems than standard max-flow on a standard computer and is an order of magnitude faster than state-of-the-art message-passing algorithms [17, 18, 19]. Our code is available at https://github.com/tajanthan/memf.

2. Preliminaries

Let X_i be a random variable taking label $x_i \in \mathcal{L}$. A pairwise MRF defined over a set of such random variables can be represented by an energy of the form

$$E(\mathbf{x}) = \sum_{i \in \mathcal{V}} \theta_i(x_i) + \sum_{(i,j) \in \mathcal{E}} \theta_{ij}(x_i, x_j)\,, \qquad (1)$$

where θ_i and θ_{ij} denote the unary potentials (*i.e.*, data costs) and pairwise potentials (*i.e.*, interaction costs), respectively. Here, $\mathcal{V}$ is the set of vertices, *e.g.*, corresponding to pixels or superpixels in an image, and $\mathcal{E}$ is the set of edges in the MRF, *e.g.*, encoding a 4-connected or 8-connected grid over the image pixels.

In this work, we consider a pairwise MRF with an ordered label set $\mathcal{L} = \{0, 1, \cdots, \ell - 1\}$, and we assume that

*NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the ARC through the ICT Centre of Excellence program.

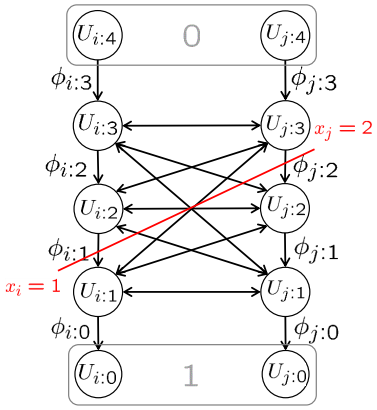

Figure 1: *Example of an Ishikawa graph. The graph incorporate edges with infinite capacity from $U_{i:\lambda}$ to $U_{i:\lambda+1}$, not shown in the graph. Here the cut corresponds to the labeling $\mathbf{x} = \{1, 2\}$ where the label set $\mathcal{L} = \{0, 1, 2, 3\}$.*

the pairwise terms are *multi-label submodular* [22]:

$$\theta_{ij}(\lambda', \mu) + \theta_{ij}(\lambda, \mu') - \theta_{ij}(\lambda, \mu) - \theta_{ij}(\lambda', \mu') \geq 0 \ , \quad (2)$$

for all $\lambda, \lambda', \mu, \mu' \in \mathcal{L}$, where $\lambda < \lambda'$ and $\mu < \mu'$. Furthermore, we assume that the pairwise potentials can be computed either by some routine or can be stored in an efficient manner. In other words, we assume that we do not need to store each individual pairwise term. Note that, in computer vision, this comes at virtually no loss of generality.

2.1. The Ishikawa graph

Ishikawa [13] introduced a method to represent the multi-label energy function (1) in a graph. The basic idea behind the Ishikawa construction is to encode the label $X_i = x_i$ of a vertex $i \in \mathcal{V}$ using binary-valued random variables $U_{i:\lambda}$, one for each $\lambda \in \{1, \cdots, \ell - 1\}$. In particular, the encoding is defined as $u_{i:\lambda} = 1$ if and only if $x_i \geq \lambda$, and 0 otherwise. The Ishikawa graph is then an st-graph $\hat{\mathcal{G}} = (\hat{\mathcal{V}} \cup \{0, 1\}, \hat{\mathcal{E}})$, consisting of one node for each $U_{i:\lambda}$, along with source and terminal nodes[1], with edges joining neighbouring nodes, as shown in Fig. 1. Note that the nodes $U_{i:\ell}$ and $U_{i:0}$ are identified with node 0 and node 1 respectively. We denote the Ishikawa edges by $e_{ij:\lambda\mu} \in \hat{\mathcal{E}}$ (contains edges in both directions) and their capacities by $\phi_{ij:\lambda\mu}$. We also denote by $e_{i:\lambda}$ the downward edge $U_{i:\lambda+1} \to U_{i:\lambda}$.

In an st-graph, a labeling $\mathbf{x}$ is represented by a "cut" in the graph (a "cut" partitions the nodes into two disjoint subsets $\hat{\mathcal{V}}_0$ and $\hat{\mathcal{V}}_1$, with $0 \in \hat{\mathcal{V}}_0$ and $1 \in \hat{\mathcal{V}}_1$). Then, the value of the energy function $E(\mathbf{x})$ is equal to the sum of the capacities on the edges from $\hat{\mathcal{V}}_0$ to $\hat{\mathcal{V}}_1$. In an Ishikawa graph, if the edge $e_{i:\lambda}$ is in the "cut", then vertex i takes label λ. Since each vertex i takes exactly one label x_i, exactly one edge $e_{i:\lambda}$ must be in the min-cut. This is ensured by having infinite capacity for each upward edge $U_{i:\lambda} \to U_{i:\lambda+1}$ in each column i.

[1] We denote them by 0 and 1, but some authors denote them by s and t.

Finding the minimum energy labeling is a min-cut problem, which can be solved optimally using max-flow [8] when the edge capacities are non-negative. As shown in [22], a *multi-label submodular* function can be represented by an Ishikawa graph with non-negative edge capacities and can therefore be minimized optimally by max-flow.

2.2. Max flow

The most popular max-flow algorithm in computer vision [5] is an augmenting path algorithm that finds a path from node 0 to node 1 through positive edges (called an *augmenting path*) and then pushes the maximum flow without exceeding the edge capacities (called *augmentation*). The augmentation operation changes the edge capacities in the graph, and therefore, the residual graph needs to be stored. That is, when applied to the Ishikawa graph, the max-flow algorithm stores $2\,\ell^2$ values per pair of neighbouring variables. For large numbers of labels and of variables, the memory requirement is high and, in many practical problems, exceeds the capacity of most computers.

2.3. Our idea

Let us assume that the max-flow algorithm is applied to the Ishikawa graph. As the algorithm proceeds, the capacities on the edges in the graph change in response to the flow. Here, instead of storing the residual graph, we propose recording the flow that has been applied to the graph.

However, since storing the flow would also require $2\,\ell^2$ values per variable pair, we propose recording two ℓ-dimensional quantities related to the flow between a pair of variables. More precisely, for each directed edge[2] $(i, j) \in \mathcal{E}^+$, we record the sum of outgoing flows from each node $U_{i:\lambda}$ to the nodes $U_{j:\mu}$ for all $\mu \in \{1, \cdots, \ell - 1\}$. We call this quantity an *exit-flow*, denoted by $\Sigma_{ij:\lambda}$ (defined below in Eq. 4). We show that these exit-flows allow us to reconstruct a *permissible* flow (defined below in Def. 3.2), which in turn lets us compute the residual edge capacities from the initial ones. Importantly, while flow reconstruction is not unique, we show that all such reconstructions are equivalent up to a *null* flow (Def. 3.3), which does not affect the energy function. Note that this idea can be applied to any augmenting path algorithm, as long as the residual graph can be rapidly constructed.

For increased efficiency, we then show how finding an augmenting path can be achieved in a simplified Ishikawa graph that amalgamates the nodes in each column into blocks. We then perform augmentation, which translates to updating our exit-flows, in this simplified graph. As a side effect, since an augmenting path in our simplified graph corresponds to a collection of augmenting paths in the Ishikawa graph, our algorithm converges in fewer iterations than the standard max-flow implementation of [5].

[2] $\mathcal{E}^+$ denotes the set of directed edges between the vertices in the MRF, *i.e.*, if $(i, j) \in \mathcal{E}$ then, $(i, j) \in \mathcal{E}^+$ and $(j, i) \in \mathcal{E}^+$.

3. Memory efficient max flow

We now introduce our memory efficient max flow algorithm, which minimizes multi-label submodular MRF energies with pairwise interactions. As mentioned in Section 2.3, our algorithm is also an augmenting path algorithm. However, instead of storing the residual graph, we propose storing exit-flows, which, at any stage of the algorithm, would allow us to compute the residual graph. In the remainder of this section, we first show how the cumulative flow can be stored in a memory efficient manner, and then turn to the problem of finding an augmenting path and performing augmentation.

3.1. Memory efficient flow encoding

Let us assume that the max-flow algorithm is applied to the Ishikawa graph. At some point in the algorithm, flow has passed along many of the edges of the graph.

Definition 3.1. A flow is a mapping $\psi : \hat{\mathcal{E}} \rightarrow \mathbb{R}$, denoted by $\psi_{ij:\lambda\mu}$ for the edges $e_{ij:\lambda\mu}$, that satisfies the anti-symmetry condition $\psi_{ij:\lambda\mu} = -\psi_{ji:\mu\lambda}$ for all $e_{ij:\lambda\mu} \in \hat{\mathcal{E}}$.

A flow is called *conservative*[3] if the total flow into a node is zero for all nodes, except for the source and the terminal, *i.e.*,

$$\sum_{j,\mu | e_{ji:\mu\lambda} \in \hat{\mathcal{E}}} \psi_{ji:\mu\lambda} = 0 \quad \forall\, U_{i:\lambda} \in \hat{\mathcal{V}}\,. \tag{3}$$

Given ψ, the residual capacities of the Ishikawa graph are updated as $\phi = \phi^0 - \psi$, where ϕ^0 represents the initial edge capacities. Furthermore, we call the flow restricted to each column *column-flows*, which we denote by $\psi_{i:\lambda}\,;\, i \in \mathcal{V}, \lambda \in \mathcal{L}$.

At first sight, it might seem that, to apply the max-flow algorithm, it is necessary to keep track of all the values $\psi_{ij:\lambda\mu}$, which would require the same order of storage as recording all the edge capacities. Below, however, we show that it is necessary to store only $\mathcal{O}(\ell)$ values for each $(i,j) \in \mathcal{E}$, instead of $\mathcal{O}(\ell^2)$.

To this end, for each $(i,j) \in \mathcal{E}^+$ and $\lambda \in \{1, \cdots, \ell-1\}$, we define an *exit-flow* as

$$\Sigma_{ij:\lambda} = \sum_{\mu} \psi_{ij:\lambda\mu}\,. \tag{4}$$

We will show that these exit-flows permit the flow ψ to be reconstructed up to equivalence.

Now, let us define some additional properties of flow, which will be useful in our exposition.

Definition 3.2. A flow ψ is called *permissible* if $\phi^0_{ij:\lambda\mu} - \psi_{ij:\lambda\mu} \geq 0$ for all $e_{ij:\lambda\mu} \in \hat{\mathcal{E}}$.

[3] A conservative flow is often referred to as a flow in the literature.

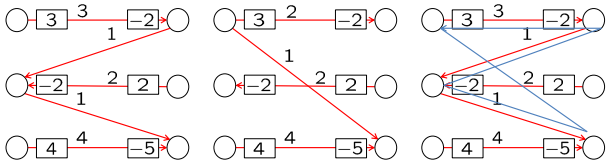

Figure 2: *An example of two equivalent flow representations with the same exit-flows. Note that each red arrow represents the value $\psi_{ij:\lambda\mu}$ and the opposite arrows $\psi_{ji:\mu\lambda}$ are not shown. Furthermore, the exit-flows Σ are shown next to the nodes and the initial edges ϕ^0 are not shown. In* ***(c)****, the flow ψ' is obtained from ψ by passing flow around a loop.*

Definition 3.3. A flow ψ is called *null* if the total flow into a node is zero for all nodes including the source and the terminal, *i.e.*, satisfies Eq. 3 for all $U_{i:\lambda} \in \hat{\mathcal{V}} \cup \{0,1\}$.

Note that a null flow does not change the energy function represented by the st-graph and it is identical to passing flow around loops. Also, if ψ is a null flow then so is $-\psi$.

Furthermore, note that the energy function encoded by an st-graph is a quadratic pseudo-boolean function [4], and a *reparametrization* of such a function is identical to a null flow in the corresponding st-graph.

Lemma 3.1. *Two sets of capacities ϕ and ϕ' represent the same energy function exactly (not up to a constant), written as $E_\phi \equiv E_{\phi'}$, if and only if $\phi' - \phi$ is a null flow.*

Proof. This lemma is a restatement of the reparametrization lemma of [17, 31] in the context of st-graphs. □

Let ϕ and ϕ' be two sets of residual capacities obtained from an initial set of capacities ϕ^0 by passing two flows ψ and ψ', *i.e.*, $\phi = \phi^0 - \psi$ and $\phi' = \phi^0 - \psi'$. If ϕ and ϕ' are equivalent, then, by Lemma 3.1, $(\phi^0 - \psi) - (\phi^0 - \psi') = \psi' - \psi$ is a null flow. Hence ψ' can be obtained from ψ by passing flow around loops in the graph. See Fig. 2.

We can now state our main theorem.

Theorem 3.1. *Let ϕ^0 be the initial capacities of an Ishikawa graph, and let Σ be a set of exit-flows. Suppose that ψ and ψ' are two flows compatible with Σ, meaning that (4) holds for both ψ and ψ', and that ψ and ψ' have identical column-flows. Then $E_{\phi^0 - \psi} \equiv E_{\phi^0 - \psi'}$.*

The idea is then as follows. If a permissible conservative flow ψ is obtained during an augmenting path flow algorithm, but only the exit-flows $\Sigma_{ij:\lambda}$ are retained for each $(i,j) \in \mathcal{E}^+$ and label λ, then one wishes, when required, to reconstruct the flow ψ on a given edge $(i,j) \in \mathcal{E}$. Although the reconstructed flow ψ' may not be identical with the flow ψ, the two will result in equivalent energy functions (not just equal up to a constant, but exactly equal for all assignments). In the augmenting path algorithm, the current flow

Algorithm 1 Flow reconstruction

Require: Given a directed edge $(i, j) \in \mathcal{E}^+$
  for $\lambda \leftarrow \ell - 1$ **to** 1 **do**
    if $\Sigma_{ij:\lambda} \geq 0$ **then**
      for $\mu \leftarrow \ell - 1$ **to** 1 **do**
        if $\Sigma_{ji:\mu} \leq 0$ **then**
          $\psi'_{ij:\lambda\mu} \leftarrow \min(|\Sigma_{ij:\lambda}|, \phi^0_{ij:\lambda\mu}, |\Sigma_{ji:\mu}|)$
          $\psi'_{ji:\mu\lambda} \leftarrow -\psi'_{ij:\lambda\mu}$
          $\Sigma_{ji:\mu} \leftarrow \Sigma_{ji:\mu} - \psi'_{ji:\mu\lambda}$
          $\Sigma_{ij:\lambda} \leftarrow \Sigma_{ij:\lambda} - \psi'_{ij:\lambda\mu}$
          if $\Sigma_{ij:\lambda} = 0$ **then**
            break

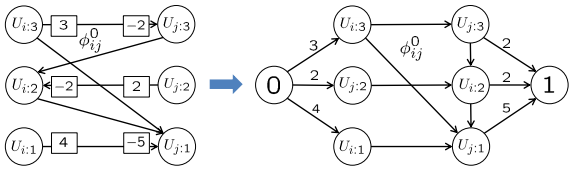

Figure 3: *Given ϕ^0 and Σ (left), flow reconstruction is formulated as a max-flow problem (right). Here the nodes with positive exit-flows are connected to the source (0) and those with negative exit-flows are connected to the terminal (1).*

values are only needed temporarily, one edge at a time, to find a new augmenting path, and hence do not need to be stored, as long as they can be rapidly computed.

Now we prove Theorem 3.1.

Proof. Given a flow ψ, let us denote its restriction to the edges $e_{ij:\lambda\mu}$ for all $\lambda, \mu \in \{1, \ldots, \ell-1\}$ for some $(i, j) \in \mathcal{E}$ by ψ_{ij}, *i.e.* restriction to cross edges only. Since both ψ_{ij} and ψ'_{ij} satisfy Eq. 4, $\psi'_{ij} - \psi_{ij}$ is a null flow. Furthermore, since both ψ and ψ' have identical column-flows, $\psi' - \psi = (\phi^0 - \psi) - (\phi^0 - \psi')$ is a null flow and, by Lemma 3.1, $E_{\phi^0-\psi} \equiv E_{\phi^0-\psi'}$. □

3.1.1 Flow reconstruction

Given the set of exit-flows Σ, the objective of the flow reconstruction problem is to find a permissible flow ψ' satisfying Eq. 4. Note that there exists a permissible conservative flow ψ compatible with Σ and hence we find ψ' such that $\psi' - \psi$ is a null flow. We do this by considering one edge $(i, j) \in \mathcal{E}$ at a time and reconstruct the flow by formulating a small max-flow problem.

Considering all the nodes $U_{i:\lambda}$ and $U_{j:\mu}$ for a given pair (i, j), we join them with edges with initial capacities $\phi^0_{ij:\lambda\mu}$. Nodes with positive exit-flow $\Sigma_{ij:\lambda}$ are joined to the source with edges of capacities $|\Sigma_{ij:\lambda}|$. Similarly, those with negative exit-flow are joined to the terminal. See Fig. 3.

Note that, in this network, the edges from the source can be thought of as "supply" and the edges to the terminal can be thought of as "demand". Since the total supply equals the total demand in this network and there exists a permissible flow ψ_{ij} compatible with Σ (*i.e.*, satisfying the supply-demand equality), the maximum flow solution of this network ψ'_{ij} is compatible with Σ, *i.e.*, satisfies Eq. 4. In fact we are interested in non-negative residual capacities $\phi'_{ij} = \phi^0_{ij} - \psi'_{ij}$ which are readily available in this network.

Now one possible max-flow algorithm is to find all the augmenting paths in this network and push maximum permissible flow through them. Note that all minimal length (length 3) augmenting paths can be found by calling Algorithm 1 twice, first for the directed edge $i \to j$ and then for $j \to i$. In our experiments such a two-pass procedure has always found a permissible flow ψ'_{ij} satisfying Eq. 4. However, in general, this may require finding longer augmenting paths, meaning that one may need to run a max-flow algorithm on this small st-graph. While this graph has $\mathcal{O}(\ell)$ nodes and $\mathcal{O}(\ell^2)$ edges, this remains perfectly tractable, since we only consider one edge (i, j) at a time. Therefore, ultimately, flow reconstruction can be done efficiently.

At this point, given the initial capacities ϕ^0 and the set of exit-flows Σ, we have shown how to reconstruct the non-negative residual edge capacities ϕ'. In fact, in addition to the set of exit-flows Σ, we need to store the column-flows $\psi_{i:\lambda}$; $i \in \mathcal{V}, \lambda \in \mathcal{L}$, to completely reconstruct the residual graph. This requires $\mathcal{O}((|\mathcal{V}| + |\mathcal{E}|)\,\ell)$ values to be stored.

3.2. Efficiently finding an augmenting path

Our algorithm follows a similar procedure as the usual max-flow, in that it iteratively finds an augmenting path and then pushes the maximum permissible flow through it. By contrast with the usual max-flow, however, we do not store the Ishikawa graph. Instead, we find an augmenting path in a simplified graph, whose construction is detailed below.

Given the capacities ϕ, we rely on the fact that there exists a label λ such that $\phi_{i:\lambda} = 0$ for each $i \in \mathcal{V}$. In fact, it is easy to see that in each column i, if all $\phi_{i:\lambda}$ are positive, then there exists a *trivial* augmenting path from $U_{i:\ell}$ to $U_{i:0}$, and the minimum along the column can be subtracted from each $\phi_{i:\lambda}$. Now, at each column i, we partition the nodes $U_{i:\lambda}$ for all $\lambda \in \{1, \cdots, \ell - 1\}$ into a set of *blocks*, such that each node in a block is connected with positive edges $e_{i:\lambda}$. Let us denote these blocks by $B_{i:\gamma}$, where γ is indexed from bottom to top starting from 0. Note that there is no edge between $B_{i:\gamma}$ and $B_{i:\gamma\pm1}$. As depicted by Fig. 4, our simplified graph then contains only the blocks and the edges between the blocks.

The edges between the blocks in the simplified graph are obtained as follows. Let us consider a directed edge $(i, j) \in \mathcal{E}^+$. We add an edge $B_{i:\gamma} \to B_{j:\delta}$, where δ is the smallest value such that $\phi_{ij:\lambda\mu}$ is positive for some $U_{i:\lambda} \in B_{i:\gamma}$ and $U_{j:\mu} \in B_{j:\delta}$. While doing this, we also enforce that there is no edge $B_{i:\gamma'} \to B_{j:\delta'}$ such that $\gamma' > \gamma$ and $\delta' < \delta$. The reasoning behind this is that, because of the upward infinite-capacity edges between the nodes $U_{i:\lambda}$ and $U_{i:\lambda+1}$, we have the following:

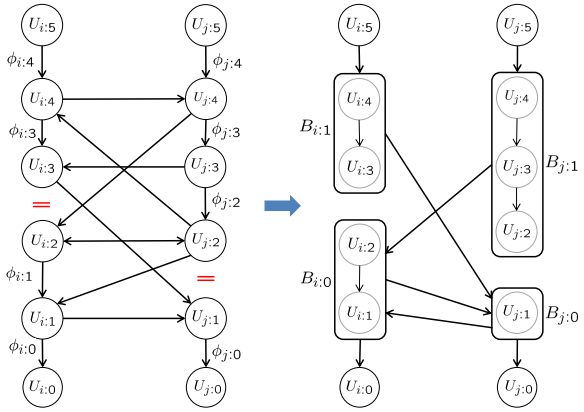

Figure 4: *To find an augmenting path in a memory efficient manner, we propose a simplified representation of the Ishikawa graph in terms of blocks corresponding to consecutive non-zero edges in each column i.*

1. If a node $U_{j:\mu}$ can be reached from $U_{i:\lambda}$ through positive edges, then the nodes $U_{j:\mu'}$, for all $\mu' \geq \mu$, can also be reached.
2. If a node $U_{j:\mu}$ can be reached from $U_{i:\lambda}$ through positive edges, then it can also be reached from the nodes $U_{i:\lambda'}$, for all $\lambda' \leq \lambda$.

Hence, an edge $B_{i:\gamma} \rightarrow B_{j:\delta}$ indicates the fact that there is some positive flow possible from any node $U_{i:\lambda} \in B_{i:\gamma'}$, for all $\gamma' \leq \gamma$, to any node $U_{j:\mu} \in B_{j:\delta'}$, for all $\delta' \geq \delta$. In other words, the set of edges obtained by this procedure is sufficient.

Now, the relationship between augmenting paths in the original Ishikawa graph and in our simplified graph can be characterized by the following theorem.

Theorem 3.2. *Given the set of Ishikawa capacities ϕ, there is an augmenting path in the simplified graph if and only if there exists an augmenting path in the Ishikawa graph.*

Proof. In supplementary material. □

Note that the simplified graph can only be used to find an augmenting path; the quantity of the maximum permissible flow cannot be determined in this graph. Therefore, the capacity of an edge $B_{i:\gamma} \rightarrow B_{j:\delta}$ is not important, but it is important to have these edges. Note also that the simplified graph is constructed incrementally for each edge $(i, j) \in \mathcal{E}$. Hence, it only requires us to store the Ishikawa edge capacities ϕ_{ij} corresponding to the edge (i, j). Furthermore, since the simplified graph $\mathcal{G}_s$ is sparse, an augmenting path can be found fairly quickly.

In addition, similar to the BK algorithm, we find an augmenting path P_s using a Breadth First Search (BFS) scheme and maintain the search tree throughout the algorithm, by repairing it whenever the simplified graph is updated. More specifically, we grow the search tree from the source (node 0), in a breadth first manner, and if the terminal (node 1) is reached, then an augmenting path is found.

3.3. Augmentation

Now, given an augmenting path P_s in the simplified graph, we want to push the maximum permissible flow through it. More specifically, since P_s corresponds to a set of augmenting paths $\{p_s\}$ in the Ishikawa graph, we will push the maximum flow through each path p_s, until no such path exists. This could be achieved by constructing the subgraph $\hat{\mathcal{G}}^p$ of the Ishikawa graph corresponding to the augmenting path P_s, and then finding each of the augmenting path p_s by searching in $\hat{\mathcal{G}}^p$. This would require us to either store $\hat{\mathcal{G}}^p$ (not memory efficient) or call the flow reconstruction algorithm too many times.

Instead, we propose breaking down the augmentation operation in the simplified graph into a sequence of flow-loops and a subtraction along a column. Then, the maximum flow through the path can be pushed in a greedy manner, by pushing the maximum flow through each flow-loop. Before describing this procedure in detail, we introduce the following definitions.

Definition 3.4. A flow-loop $m(\lambda, \mu, \alpha)$ in the Ishikawa graph is defined as the following sequence of operations: First, a value α is pushed down the left column from $U_{i:\ell}$ to $U_{i:\lambda}$, then across from $U_{i:\lambda}$ to $U_{j:\mu}$, and finally up the right column from $U_{j:\mu}$ to $U_{j:\ell}$. Thus, applying the flow-loop $m(\lambda, \mu, \alpha)$ corresponds to replacing ϕ by $\phi + \Delta$, where

$$\begin{aligned} \Delta_{i:\lambda'} &= -\alpha \quad \forall \lambda' \geq \lambda \,, \\ \Delta_{ij:\lambda\mu} &= -\alpha \,, \\ \Delta_{ji:\mu\lambda} &= \alpha \,, \\ \Delta_{j:\mu'} &= \alpha \quad \forall \mu' \geq \mu \,. \end{aligned}$$

Definition 3.5. A flow-loop $\tilde{m}(\gamma, \delta, \alpha)$ in the simplified graph $\mathcal{G}_s$ is defined by the following sequence of operations: First a value α is pushed down the left column from $U_{i:\ell}$ to $B_{i:\gamma}$, then across from $B_{i:\gamma}$ to $B_{j:\delta}$, and finally up the right column from $B_{j:\delta}$ to $U_{j:\ell}$.

Note that, for a flow-loop $\tilde{m}(\gamma, \delta, \alpha)$ to be permissible, block $B_{i:\gamma}$ must contain node $U_{i:\ell-1}$. Note also that the flow-loop $\tilde{m}(\gamma, \delta, \alpha)$ can be thought of as a summation of flow-loops $m(\lambda, \mu, \alpha')$, where $U_{i:\lambda} \in B_{i:\gamma}$ and $U_{j:\mu} \in B_{j:\delta'}$, for all $\delta' \geq \delta$ (see Fig. 5).

Given these definitions, one can easily see that the augmentation operation along the path P_s can be broken down into a sequence of flow-loops $\tilde{m}(\gamma, \delta, \alpha)$ and a subtraction along the last column k, as illustrated in Fig. 6. Now, we push the maximum permissible flow through P_s, using the following greedy approach.

For each edge $B_{i:\gamma} \rightarrow B_{j:\delta}$ that is part of the path P_s, we apply a flow-loop $\tilde{m}(\gamma, \delta, \alpha_{ij})$, where α_{ij} is the maximum permissible flow through the edge $B_{i:\gamma} \rightarrow B_{j:\delta}$. In fact, applying this flow-loop translates to reconstructing the

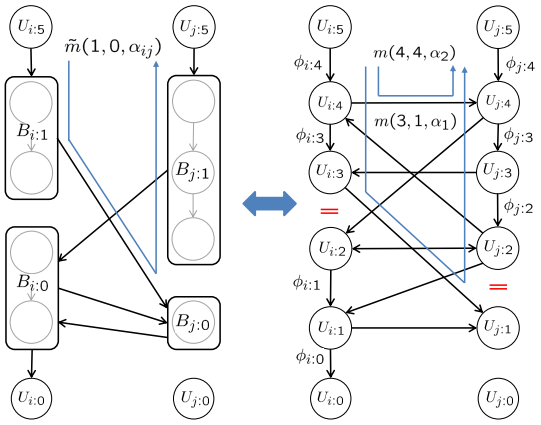

Figure 5: *An example flow-loop $\tilde{m}(1,0,\alpha_{ij})$ in the simplified graph (left) is equivalent to the summation of two flow-loops $m(3,1,\alpha_1)$ and $m(4,4,\alpha_2)$ in the Ishikawa graph (right), with $\alpha_{ij} = \alpha_1 + \alpha_2$.*

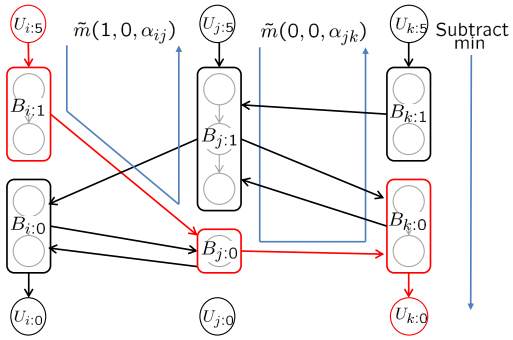

Figure 6: *An augmentation operation is broken down into a sequence of flow-loops $\tilde{m}(\gamma,\delta,\alpha)$, and a subtraction along the column k. The augmenting path P_s is highlighted in red.*

Ishikawa edge capacities ϕ_{ij} corresponding to edge (i,j) and then applying flow-loops $m(\lambda,\mu,\alpha')$ for all $\lambda \geq \check{\lambda}$ and $\mu \geq \check{\mu}$, starting from $\check{\lambda}$ and $\check{\mu}$, until no permissible flow-loop $m(\lambda,\mu,\alpha')$ exists, with $\check{\lambda}$ and $\check{\mu}$ the smallest values such that $U_{i:\lambda} \in B_{i:\gamma}$ and $U_{j:\mu} \in B_{j:\delta}$. Finally, in the last column k, all the values $\phi_{k:\lambda}$ are positive, and the minimum along column k is subtracted from each $\phi_{k:\lambda}$. It is easy to see that this approach pushes the maximum permissible flow through the path P_s.

Since, for each edge (i,j), we do not store all the $2\,\ell^2$ capacities, but only the $2\,\ell$ exit-flows Σ, augmentation must then also update these values. Fortunately, there is a direct relation between the flow-loops and Σ. To see this, let us consider the example flow-loop $\tilde{m}(1,0,\alpha_{ij})$ shown in Fig. 5. Applying this flow-loop updates the corresponding exit-flows as

$$\begin{aligned} \Sigma_{ij:3} &= \Sigma_{ij:3} + \alpha_1 \,, \\ \Sigma_{ji:1} &= \Sigma_{ji:1} - \alpha_1 \,, \\ \Sigma_{ij:4} &= \Sigma_{ij:4} + \alpha_2 \,, \\ \Sigma_{ji:4} &= \Sigma_{ji:4} - \alpha_2 \,. \end{aligned} \tag{5}$$

Algorithm 2 Memory Efficient Max Flow (MEMF)

Require: ϕ^0 ▷ Initial Ishikawa capacities
  $\Sigma \leftarrow 0, T \leftarrow \emptyset$ ▷ Initialize exit-flows and search tree
  $\mathcal{G}_s \leftarrow$ simplify_graph(ϕ^0) ▷ Initial simplified graph
  repeat
    $(T, P_s) \leftarrow$ augmenting_path$(\mathcal{G}_s, T)$ ▷ Sec. 3.2
    $\Sigma \leftarrow$ augment(P_s, ϕ^0, Σ) ▷ Sec. 3.3
    for each edge $(i,j) \in \mathcal{E}$ affected by augmentation **do**
      $\phi_{ij} \leftarrow$ compute_edges(ϕ^0, Σ, i, j) ▷ Sec. 3.1.1
      $\mathcal{G}_s^{ij} \leftarrow$ simplify_graph(ϕ_{ij}, i, j) ▷ Sec. 3.2
    $T \leftarrow$ repair_tree$(T, \mathcal{G}_s)$ ▷ Repair search tree
  until no augmenting paths possible
  return get_labelling(T) ▷ Read from search tree

Similar updates can be done for all flow-loops in our procedure. Note that the edge $B_{i:\gamma} \rightarrow B_{j:\delta}$ represents a collection of possible paths from all the nodes $U_{i:\lambda} \in B_{i:\gamma}$ to all the nodes $U_{j:\mu} \in B_{j:\delta'}$, for all $\delta' \geq \delta$. Therefore, unlike in the full Ishikawa graph, after applying a flow-loop, the portion of the graph $\mathcal{G}_s^{ij}$ corresponding to edge $(i,j) \in \mathcal{E}$ needs to be reconstructed. This, however, can be done in a memory efficient manner, since it only involves one edge (i,j) at a time.

3.4. Summary

Our memory efficient max-flow (**MEMF**) method is summarized in Algorithm 2. Let us briefly explain the subroutines below.

simplify_graph: Given the initial Ishikawa capacities ϕ^0, this subroutine constructs the simplified graph by amalgamating nodes into blocks as described in Section 3.2. If the input to the subroutine is the Ishikawa capacities ϕ_{ij} corresponding to the edge $(i,j) \in \mathcal{E}$, then it constructs the simplified graph portion $\mathcal{G}_s^{ij}$.

augmenting_path: Given the simplified graph $\mathcal{G}_s$ and the search tree T, this subroutine finds an augmenting path P_s by growing the search tree, as discussed in Section 3.2.

augment: Given the path P_s, this subroutine pushes the maximum permissible flow through it by applying flow-loops $\tilde{m}(\gamma,\delta,\alpha)$ and then subtracting the minimum from the last column, as discussed in Section 3.3.

compute_edges: Given the initial Ishikawa edge capacities ϕ^0 and the set of exit-flows Σ, this subroutine computes the non-negative residual Ishikawa edge capacities ϕ_{ij} corresponding to the given edge (i,j). This is accomplished by solving a small max-flow problem (see Section 3.1.1).

repair_tree: This subroutine is similar to the *adoption* stage of the BK algorithm. Given the reconstructed simplified graph, the search tree T is repaired by checking for valid *parents* for each *orphan* node. See Section 3.2.3 in [5] for more details.

get_labelling: This subroutine directly reads the optimal labelling from the search tree T.

As discussed above, the exit-flows Σ require $\mathcal{O}(\ell)$ storage for each edge $(i,j) \in \mathcal{E}$. In addition, the simplified graph $\mathcal{G}_s$ can have at most $\mathcal{O}(|\mathcal{V}|\,\ell)$ blocks and $\mathcal{O}(|\mathcal{E}|\,\ell)$ edges. Furthermore, recall that we assume that the initial Ishikawa capacities ϕ^0 can be stored efficiently. Therefore, ultimately, our algorithm requires $\mathcal{O}((|\mathcal{V}|+|\mathcal{E}|)\,\ell)$ values to be stored.

Note that, even though our algorithm is efficient, similarly to the BK algorithm, it lacks a polynomial time guarantee. In fact, we lose the ability to find the shortest augmenting path in the Ishikawa graph, due to graph simplification. Therefore, it would be interesting to come up with a simplification strategy that can yield a polynomial time bound on our algorithm.

4. Related work

The approaches that have been proposed to minimize multi-label submodular MRFs can be roughly grouped into two categories: Those based on max-flow and those based on an LP relaxation of the problem. Below, we briefly review representative techniques in each category.

Max-flow-based methods. The most popular method to minimize a multi-label submodular MRF energy is to construct the Ishikawa graph [13] and then apply a max-flow algorithm to find the min-cut solution. Broadly speaking, there are three different kinds of max-flow algorithms: those relying on finding augmenting paths [8], the push-relabel approach [12] and the pseudo-flow techniques [6]. Even though numerous implementations are available, the BK method [5] is arguably the fastest implementation for 2D and sparse 3D graphs. Recently, for dense problems, the IBFS algorithm [11] was shown to outperform the BK method in a number of experiments [28]. All the above-mentioned algorithms, however, require the same order of storage as the Ishikawa graph and hence scale poorly. Two approaches have nonetheless been studied to scale the max-flow algorithms. The first one explicitly relies on the N-D grid structure of the problem at hand [7, 14]. The second one makes use of distributed computing [23, 24, 29]. Unfortunately, both these approaches require additional resources (disk space or clusters) to run max-flow on an Ishikawa graph. By contrast, our algorithm lets us efficiently minimize the energy of much larger Ishikawa-type graphs on a standard computer. Furthermore, using the method of [24], it can also be parallelized.

LP relaxation-based methods. One memory-efficient way to minimize a multi-label submodular MRF energy consists of formulating the problem as a linear program and then maximize the dual using message-passing techniques [30]. Many such algorithms have been studied [17, 18, 19, 31]. Even though these algorithms are good at approximating the optimal solution, as evidenced by the comparison of [15] and by our experiments, they usually take much longer to converge to the optimal solution than max-flow-based techniques.

Figure 7: *Left and right images of the stereo instance from the KITTI dataset. The images are of size* 1241×376*, and we set the number of labels to 40. This image pair was chosen arbitrarily as a representative of the dataset.*

5. Experiments

We evaluated our algorithm on the problems of stereo correspondence estimation and image inpainting. For stereo correspondence estimation, we employed six instances from the Middlebury dataset [20, 21]: Tsukuba, Venus, Sawtooth, Map, Cones and Teddy, and one instance from the KITTI dataset [9] (see Fig. 7). For Tsukuba and Venus, we used the unary potentials of [25], and for all other stereo cases, those of [3]. For inpainting, we used the Penguin and House images employed in [25], and we used the same unary potentials as in [25]. In all the above cases, we used pairwise potentials that can be expressed as

$$\theta_{ij}(x_i, x_j) = w_{ij}\,\theta(|x_i - x_j|)\;, \tag{6}$$

where, unless stated otherwise, the regularizer $\theta(|x_i - x_j|)$ is the quadratic function. Furthermore, in all our experiments, we employed a 4-connected neighbourhood.

We compare our results with two max-flow implementations: the BK method [5] and Excesses Incremental Breadth First Search (EIBFS) [10], and three LP relaxation-based algorithms: Tree Reweighted Message Passing (TRWS) [17], Subgradient based Dual Decomposition (DDSG) [18] and the Adaptive Diminishing Smoothing algorithm (ADSal) [19]. For DDSG and ADSal, we used the Opengm [2] implementations. For the other algorithms, we employed the respective authors' implementations.

In practice, we only ran the BK method and EIBFS if the graph could be stored in RAM. Otherwise, we provide an estimate of their memory requirement. For LP relaxation-based methods, unless they converged, we ran the algorithms either for 10000 iterations, or for 50000 seconds, whichever occurred first. Note that the running times reported for our algorithm include graph construction. All our experiments were conducted on a 3.4 GHz i7-4770 CPU with 16 GB RAM.

The memory consumption and running times of the algorithms are provided in Table 1. Altogether, our algorithm lets us solve much larger problems than the BK method and EIBFS, and is an order of magnitude faster than state-of-the-art message-passing algorithms.

Problem	Memory [MB]						Time [s]					
	BK	EIBFS	DDSG	ADSal	TRWS	MEMF	BK	EIBFS	DDSG	ADSal	TRWS	MEMF
Tsukuba	3195	2495	258	252	287	**211**	14	**4**	>9083	>7065	198	28
Venus	7626	5907	424	418	638	**396**	35	**9**	>18156	1884	206	59
Sawtooth	7566	5860	415	415	633	**393**	31	**8**	>16238	10478	455	35
Map	6454	4946	**171**	208	494	219	57	**9**	>9495	>1679	187	36
Cones	*72303	*55063	**657**	939	5024	1200	-	-	>50000	>17866	1095	**364**
Teddy	*72303	*55063	**659**	939	5025	1200	-	-	>50000	>50000	6766	**2055**
KITTI	*88413	*67316	**1422**	1802	6416	2215	-	-	>50000	>50000	>45408	**18665**
Penguin	*173893	*130728	236	1123	**215**	663	-	-	>50000	>50000	>50000	**6504**
House	*521853	*392315	689	2389	**643**	1986	-	-	>50000	>50000	>50000	**9001**

Table 1: *Memory consumption and runtime comparison with state-of-the-art baselines. A "*" indicates a memory estimate, and ">" indicates that the algorithm did not converge to the optimum within the specified time. Note that our algorithm has a memory consumption $\mathcal{O}(\ell)$ times lower than the max-flow-based methods and is an order of magnitude faster than message-passing algorithms. Compared to EIBFS, our algorithm is only 4 – 7 times slower, but requires 12 – 23 times less memory, which makes it applicable to more realistic problems. In all stereo problems, TRWS cached the pairwise potentials in an array for faster retrieval, but in the case of inpainting, it was not possible due to excessive memory requirement.*

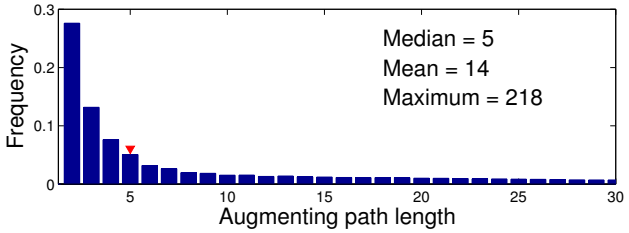

Figure 8: *Lengths of augmenting paths found by our algorithm for the Tsukuba stereo instance. Each bar indicates the proportion of paths of a certain length. For example, out of all augmenting paths 28% of them were of length 2. The red arrow indicates the median length.*

5.1. MEMF analysis

Note that, at each iteration, *i.e.*, at each augmentation step, our algorithm performs more computation than standard max-flow. Therefore, we would like our algorithm to find short augmenting paths and to converge in fewer iterations than standard max-flow. Below, we analyze these two properties empirically.

In Fig. 8, we show the distribution of the lengths of the augmenting paths found by our algorithm for the Tsukuba stereo instance. Note that the median length is only 5. As a matter of fact, the maximum length observed over all our experiments was 1073 for the KITTI data. Nevertheless, even in that image, the median length was only 15. Note that, since our algorithm finds augmenting paths in a simplified graph, the path lengths are not directly comparable to those found by other max-flow-based methods. In terms of number of augmentations, we found that our algorithm only required between 35% and 50% of the total number of augmentations of the BK method.

5.2. Minimizing non-submodular MRFs

Since our algorithm can simply replace standard max-flow in Ishikawa-type graphs, we replaced the BK method with our MEMF procedure in the IRGC algorithm [1], which minimizes MRFs with some non-convex pairwise potentials by iteratively building and solving an Ishikawa graph. This lets us tackle much larger non-submodular problems. In particular, we computed inpainting results on Penguin by using all 256 labels, as opposed to the down-sampled label sets used in [1]. The results of the IRGC+expansion algorithm, with the BK method (IRGC) and with MEMF (MIRGC) are summarized in Table 2.

Problem	Memory [MB]		Time [s]	
	IRGC	MIRGC	IRGC	MIRGC
Penguin-128/10	4471	**332**	**224**	2566
House-64/15	8877	**498**	**106**	409
Penguin-256/20	*17143	**663**	-	**17748**
House-256/60	*137248	**1986**	-	**19681**

Table 2: *Memory consumption and runtime comparison of IRGC+expansion with either the BK method or our MEMF algorithm as subroutine (IRGC and MIRGC respectively). Here, "Penguin-128/10" corresponds to the Penguin problem with 128 labels and the truncated quadratic function with truncation value 10 as pairwise potential. A "*" indicates a memory estimate. Compared to IRGC, MIRGC is only 4 – 11 times slower but requires 13 – 18 times less memory, which makes it applicable to much larger MRFs.*

6. Conclusion

We have introduced a variant of the max-flow algorithm that can minimize multi-label submodular MRF energies optimally, while requiring much less storage. Furthermore, our experiments have shown that our algorithm is an order of magnitude faster than state-of-the-art methods. We therefore believe that our algorithm constitutes the method of choice to minimize Ishikwa-type graphs when the complete graph cannot be stored in memory.

References

[1] T. Ajanthan, R. Hartley, M. Salzmann, and H. Li. Iteratively reweighted graph cut for multi-label mrfs with non-convex priors. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. 1, 8

[2] B. Andres, T. Beier, and J. H. Kappes. Opengm: A c++ library for discrete graphical models. *arXiv preprint arXiv:1206.0111*, 2012. 7

[3] S. Birchfield and C. Tomasi. A pixel dissimilarity measure that is insensitive to image sampling. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(4):401–406, 1998. 7

[4] E. Boros and P. L. Hammer. Pseudo-boolean optimization. *Discrete applied mathematics*, 123(1):155–225, 2002. 3

[5] 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. 1, 2, 6, 7

[6] B. G. Chandran and D. S. Hochbaum. A computational study of the pseudoflow and push-relabel algorithms for the maximum flow problem. *Operations research*, 57(2):358–376, 2009. 7

[7] A. Delong and Y. Boykov. A scalable graph-cut algorithm for nd grids. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008. 7

[8] L. Ford and D. R. Fulkerson. *Flows in networks*, volume 1962. Princeton Princeton University Press, 1962. 1, 2, 7

[9] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, page 0278364913491297, 2013. 7

[10] A. V. Goldberg, S. Hed, H. Kaplan, P. Kohli, R. E. Tarjan, and R. F. Werneck. Faster and more dynamic maximum flow by incremental breadth-first search. In *Algorithms–ESA 2015*, pages 619–630. Springer, 2015. 7

[11] A. V. Goldberg, S. Hed, H. Kaplan, R. E. Tarjan, and R. F. Werneck. Maximum flows by incremental breadth-first search. In *Algorithms–ESA 2011*, pages 457–468. Springer, 2011. 7

[12] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940, 1988. 7

[13] H. Ishikawa. Exact optimization for markov random fields with convex priors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(10):1333–1336, 2003. 1, 2, 7

[14] O. Jamriška, D. Sỳkora, and A. Hornung. Cache-efficient graph cuts on structured grids. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3673–3680. IEEE, 2012. 7

[15] J. H. Kappes, B. Andres, F. A. Hamprecht, C. Schnörr, S. Nowozin, D. Batra, S. Kim, B. X. Kausler, T. Kröger, J. Lellmann, N. Komodakis, B. Savchynskyy, and C. Rother. A comparative study of modern inference techniques for structured discrete energy minimization problems. *International Journal of Computer Vision*, pages 1–30, 2015. 7

[16] P. Kohli and P. H. Torr. Efficiently solving dynamic markov random fields using graph cuts. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 922–929. IEEE, 2005. 1

[17] V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(10):1568–1583, 2006. 1, 3, 7

[18] N. Komodakis, N. Paragios, and G. Tziritas. Mrf energy minimization and beyond via dual decomposition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(3):531–552, 2011. 1, 7

[19] B. Savchynskyy, S. Schmidt, J. H. Kappes, and C. Schnörr. Efficient mrf energy minimization via adaptive diminishing smoothing. In *Uncertainty in Artificial Intelligence*, pages 746–755, 2012. 1, 7

[20] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 47(1-3):7–42, 2002. 7

[21] D. Scharstein and R. Szeliski. High-accuracy stereo depth maps using structured light. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 1, pages I–195. IEEE, 2003. 7

[22] D. Schlesinger and B. Flach. *Transforming an arbitrary min-sum problem into a binary one*. TU, Fak. Informatik, 2006. 1, 2

[23] A. Shekhovtsov and V. Hlaváč. A distributed min-cut/maxflow algorithm combining path augmentation and push-relabel. *International journal of computer vision*, 104(3):315–342, 2013. 7

[24] P. Strandmark and F. Kahl. Parallel and distributed graph cuts by dual decomposition. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2085–2092. IEEE, 2010. 1, 7

[25] 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 with smoothness-based priors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(6):1068–1080, 2008. 7

[26] P. Torr and M. Kumar. Improved moves for truncated convex models. In *Advances in neural information processing systems*, pages 889–896, 2009. 1

[27] O. Veksler. Multi-label moves for mrfs with truncated convex priors. *International journal of computer vision*, 98(1):1–14, 2012. 1

[28] T. Verma and D. Batra. Maxflow revisited: An empirical comparison of maxflow algorithms for dense vision problems. In *BMVC*, pages 1–12, 2012. 7

[29] V. Vineet and P. Narayanan. Cuda cuts: Fast graph cuts on the gpu. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*, pages 1–8. IEEE, 2008. 7

[30] 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. 7

[31] T. Werner. A linear programming approach to max-sum problem: A review. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(7):1165–1179, 2007. 3, 7