Memory Efficient Max Flow for Multi-label Submodular MRFs

Thalaiyasingam Ajanthan, *Student Member, IEEE*, Richard Hartley, *Fellow, IEEE*, and Mathieu Salzmann, *Member, IEEE*

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.

Index Terms—Max-flow, Mutli-label submodular, Memory efficiency, Flow encoding, Graphical models.

---------- ◆ ----------

1 INTRODUCTION

Ishikawa [1] introduced a max-flow-based method to globally minimize the energy of multi-label MRFs with convex edge terms. In [2], 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 [3], 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.

- T. Ajanthan and R. Hartley are with the College of Engineering and Computer Science, Australian National University, Canberra and Data61, CSIRO, Canberra.
  E-mail: thalaiyasingam.ajanthan@anu.edu.au
- M. Salzmann is with the Computer Vision Laboratory, École Polytechnique Fédérale de Lausanne.
- Data61 (formerly NICTA) is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council (ARC) through the ICT Centre of Excellence program.
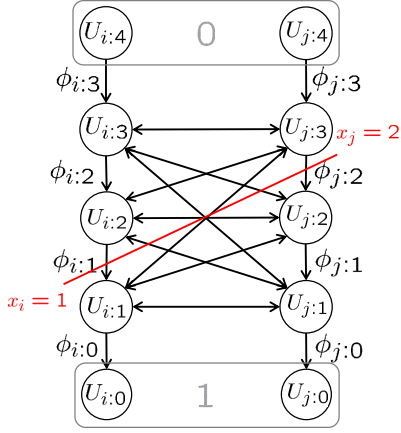
The optimality of Ishikawa's formalism made it a method of choice as a subroutine in many approximate energy minimization algorithms, such as multi-label moves [4], [5] and IRGC [6]. Since our approach can simply replace the standard max-flow algorithm [7] in Ishikawa-type graphs, it also allows us to 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 [8] and also be accelerated using the parallel max-flow technique [9].

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 [10], [11], [12]. Our code is available at https://github.com/tajanthan/memf.

A preliminary version of this paper is appeared in [13]. This extended version contains a polynomial time version of the MEMF algorithm, a discussion on the equivalence with min-sum message passing and an experiment to evaluate the empirical time complexity.

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.

Fig. 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 $\mathcal{L} = \{0, 1, 2, 3\}$.*

In this work, we consider a pairwise MRF with an ordered label set $\mathcal{L} = \{0, 1, \cdots, \ell - 1\}$, and we assume that the pairwise terms satisfy the *multi-label submodularity* condition [2]:

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

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 [1] 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 label $\lambda \in \mathcal{L}$. 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 $\hat{\mathcal{G}} = (\hat{\mathcal{V}} \cup \{0, 1\}, \hat{\mathcal{E}})$[1], where the set of nodes and the set of edges are defined as follows:

$$\hat{\mathcal{V}} = \{U_{i:\lambda} \mid i \in \mathcal{V}, \lambda \in \{1, \cdots, \ell - 1\}\} , \quad (3)$$
$$\hat{\mathcal{E}} = \hat{\mathcal{E}}_v \cup \hat{\mathcal{E}}_c ,$$
$$\hat{\mathcal{E}}_v = \{(U_{i:\lambda}, U_{i:\lambda \pm 1}) \mid i \in \mathcal{V}, \lambda \in \{1, \cdots, \ell - 1\}\} ,$$
$$\hat{\mathcal{E}}_c = \{(U_{i:\lambda}, U_{j:\mu}), (U_{j:\mu}, U_{i:\lambda}) \mid (i, j) \in \mathcal{E}, U_{i:\lambda}, U_{j:\mu} \in \hat{\mathcal{V}}\} ,$$

where $\hat{\mathcal{E}}_v$ is the set of vertical edges and $\hat{\mathcal{E}}_c$ is the set of cross edges. Note that, the nodes $U_{i:\ell}$ and $U_{i:0}$ are identified as 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 $(U_{i:\lambda+1}, U_{i:\lambda})$. An example of an Ishikawa graph is shown in Fig. 1.

In an *st*-graph, a labeling $\mathbf{x}$ is represented by a "cut" in the graph (a "cut" partitions the nodes in the graph 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 downward edge $e_{i:\lambda}$ is in the "cut", then vertex i takes label λ. In MRF energy minimization, each vertex i takes exactly one label x_i, which means that exactly one edge $e_{i:\lambda}$ must be in the min-cut of the Ishikawa graph. This is ensured by having infinite capacity for each upward edge $e_{ii:\lambda\lambda+1}$, *i.e.*, $\phi_{ii:\lambda\lambda+1} = \infty$ for all $i \in \mathcal{V}$ and $\lambda \in \mathcal{L}$. Note that, by construction of the Ishikawa graph, the capacities ϕ and the energy parameters θ are related according to the following formula:

$$\theta_i(\lambda) = \phi_{ii:\lambda+1\lambda} = \phi_{i:\lambda} , \quad (4)$$
$$\theta_{ij}(\lambda, \mu) = \sum_{\substack{\lambda' > \lambda \\ \mu' \leq \mu}} \phi_{ij:\lambda'\mu'} + \sum_{\substack{\lambda' \leq \lambda \\ \mu' > \mu}} \phi_{ji:\mu'\lambda'} .$$

Finding the minimum energy labeling is a min-cut problem, which can be solved optimally using the max-flow algorithm [3] when the edge capacities are non-negative. As shown in [2], a *multi-label submodular* energy function can be represented by an Ishikawa graph with non-negative edge capacities ϕ and can therefore be minimized optimally by max-flow.

2.2 Max Flow

The most popular max-flow algorithm in computer vision [7] 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 number 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 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. 6). We show that these exit-flows allow us to reconstruct a *permissible* flow (defined below in Def. 3.3), 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. 3.4), which does not affect the energy function. Note that this idea can be applied to

---

1. Some authors denote the nodes 0 and 1 by s and t.

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 $(j, i) \in \mathcal{E}^+$.

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 (called *block-graph*) that amalgamates the nodes in each column into blocks. We then perform augmentation, which translates to updating our exit-flows, in this block-graph. As a side effect, since an augmenting path in our block-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 [7].

3 MEMORY EFFICIENT FLOW ENCODING

Before we introduce our memory efficient max flow algorithm, let us describe how the cumulative flow can be stored in a memory efficient manner. This technique can be used in any augmenting path flow algorithm, by reconstructing the residual edge capacities whenever needed.

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 $\psi : \hat{\mathcal{E}} \to \mathbb{R}$, 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.*,

$$\sum_{j,\mu|e_{ji:\mu\lambda}\in\hat{\mathcal{E}}} \psi_{ji:\mu\lambda} = 0 \quad \forall U_{i:\lambda} \in \hat{\mathcal{V}} . \tag{5}$$

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, the flow values that we store in our algorithm, namely *source-flows* and *exit-flows* are defined below.

Definition 3.2. 1) For each $i \in \mathcal{V}$, the flow out from the source node $\psi_{i:\ell-1}$ is called a *source-flow*.
2) For each $(i, j) \in \mathcal{E}^+$ and $\lambda \in \{1, \cdots, \ell-1\}$, we define an *exit-flow* as

$$\Sigma_{ij:\lambda} = \sum_{\mu} \psi_{ij:\lambda\mu} . \tag{6}$$

We will show that these source-flows and 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 3.3. 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.
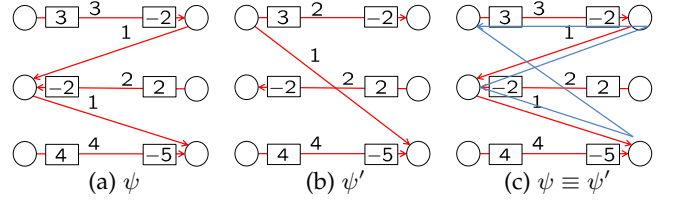

Fig. 2: *An example of two equivalent flow representations with the same exit-flows. Note that each red arrow represents the value $\psi_{ij:\lambda\mu}$ and the opposite arrows $\psi_{ji:\mu\lambda}$ are not shown. Furthermore, the exit-flows Σ are shown next to the nodes and the initial edges ϕ^0 are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

Definition 3.4. 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. 5 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 [14], 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 [10], [15] 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 conservative flows compatible with Σ, meaning that (6) holds for both ψ and ψ', and that ψ and ψ' have identical source-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 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. First we will prove that ψ and ψ' have identical column-flows. For a conservative flow

$$\psi_{i:\lambda} - \left(\psi_{i:\lambda-1} + \sum_{(i,j)\in\mathcal{E}^+} \Sigma_{ij:\lambda} \right) = 0 , \tag{7}$$
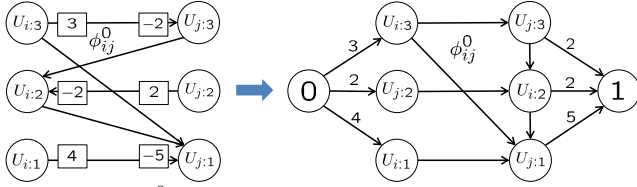
Fig. 3: *Given ϕ^0 and Σ (left), flow reconstruction is formulated as a max-flow problem (right). Here the nodes with positive exit-flows are connected to the source (0) and those with negative exit-flows are connected to the terminal (1).*

for all $i \in \mathcal{V}$ and $\lambda \in \{1, \ldots, \ell - 1\}$. Since ψ and ψ' are compatible with Σ and have identical source-flows, $\psi_{i:\lambda} = \psi'_{i:\lambda}$ for all $i \in \mathcal{V}$ and $\lambda = \mathcal{L}$. Hence they have identical column-flows.

Now we will prove the equivalence. 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. 6, $\psi'_{ij} - \psi_{ij}$ 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 Flow Reconstruction

Note that, from Eq. 7 it is clear that given the source-flows $\psi_{i:\ell-1}$; $i \in \mathcal{V}$, the column-flows $\psi_{i:\lambda}$; $i \in \mathcal{V}, \lambda \in \mathcal{L}$ can be computed in a top-down fashion. Now we turn to the problem of finding the flows along the cross-edges $e_{ij:\lambda\mu}$.

Given the set of exit-flows Σ, the objective is to find a permissible flow ψ' satisfying Eq. 6. 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. 6. In fact we are interested in non-negative residual capacities $\phi'_{ij} = \phi^0_{ij} - \psi'_{ij}$ which are readily available in this network.

This problem can be solved using a greedy augmenting path algorithm. 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, the source-flows $\psi_{i:\ell-1}$; $i \in \mathcal{V}$ and the set of exit-flows Σ, we have shown how to reconstruct the non-negative residual edge capacities ϕ'. This requires $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}| \ell)$ values to be stored.

4 POLYNOMIAL TIME MEMORY EFFICIENT MAX FLOW

We now introduce our polynomial time memory efficient max flow algorithm, which minimizes multi-label submodu-

lar MRF energies with pairwise interactions. Our algorithm follows a similar procedure as the standard Edmonds-Karp algorithm [16], in that it iteratively finds the shortest augmenting path and then pushes the maximum flow through it without exceeding the edge capacities. However, instead of storing the residual graph, we store exit-flows as proposed in Section 3, which, at any stage of the algorithm, would allow us to compute the residual graph. Below, we discuss how one can find an augmenting path and update the exit-flows, *i.e.*, perform augmentation, without storing the full Ishikawa graph.

4.1 Finding an Augmenting Path

Our algorithm finds an augmenting path in a subgraph of the Ishikawa graph, called *lower-graph*. In particular, the lower-graph contains only a subset of Ishikawa edges which satisfy the *lowest-cross-edge* property.

Definition 4.1. Consider a directed edge $(i, j) \in \mathcal{E}^+$. For each node $U_{i:\lambda}$, the *lowest-cross-edge* is defined as, the edge $e_{ij:\lambda\mu}$ where μ is the smallest value such that $\phi_{ij:\lambda\mu} > 0$.

More specifically, in addition to the vertical edges $\hat{\mathcal{E}}_v$, the lower-graph contains the lowest-cross-edges. Therefore, we only store $\mathcal{O}(\ell)$ edges per variable pair (i, j). Now, the relationship between augmenting paths in the original Ishikawa graph and the lower-graph can be characterized by the following theorem.

Theorem 4.1. *Given the Ishikawa graph, there is an augmenting path in the lower-graph if and only if there exists an augmenting path in the Ishikawa graph.*

Proof. Since the lower-graph is a subgraph of the Ishikawa graph, if there is an augmenting path in the lower-graph, then there exists an augmenting path in the Ishikawa graph.

We will now prove the converse. Consider a directed edge $(i, j) \in \mathcal{E}^+$. Let $e_{ij:\lambda\mu}$ and $e_{ij:\lambda\mu'}$ be two positive capacity edges from $U_{i:\lambda}$ and $e_{ij:\lambda\mu'}$ be the lowest-cross-edge. Then, due to the upward infinite capacity edges from $U_{j:\mu'} \leadsto U_{j:\mu}$, there is a positive capacity path from $U_{i:\lambda} \leadsto U_{j:\mu}$ through the lowest-cross-edge $e_{ij:\lambda\mu'}$. This proves the theorem. □

This enables us to find all the augmenting paths in the Ishikawa graph by searching in a smaller graph that has $\mathcal{O}(\ell)$ edges per variable pair (i, j).

Note that, as mentioned earlier, we find the *shortest* augmenting path in this lower-graph. However, by contrast to the Edmonds-Karp algorithm [16], the path distance is computed considering *zero* distance for the infinite capacity edges and unit distance for other edges, instead of unit distance for all the edges. The intuition for this modification is that, the infinite capacity edges will never become saturated (or eliminated from the graph) for the entire course of the algorithm. Note that, with this definition of path distance, the augmenting paths in both lower-graph and Ishikawa graph have same length. This will enable us to prove the polynomial time bound of our algorithm in a similar manner as the standard Edmonds-Karp algorithm. Note that, even in this case, the shortest augmenting path can be found using a Breadth First Search (BFS) scheme.

Algorithm 1 Memory Efficient Max Flow (MEMF) - Polynomial Time Version

---

Require: ϕ^0 ▷ Initial Ishikawa capacities
  $\Sigma \leftarrow 0$ ▷ Initialize exit-flows
  $\bar{\phi} \leftarrow$ lower-graph(ϕ^0)   ▷ Store the lowest-cross-edges
  repeat
      $p \leftarrow$ shortest_augmenting_path($\bar{\phi}$)   ▷ Sec. 4.1
      $(\bar{\phi}, \Sigma) \leftarrow$ augment($p, \bar{\phi}$)   ▷ Sec. 4.2
      for each edge $e_{ij:\lambda\mu}$ becomes saturated **do**
         $\phi_{ij} \leftarrow$ compute_edges(ϕ^0, Σ, i, j)   ▷ Sec. 3.1
         $\bar{\phi}_{ij} \leftarrow$ lower-graph(ϕ_{ij}, i, j)   ▷ Sec. 4.1
  until no augmenting paths possible
  return get_labelling($\bar{\phi}$)   ▷ Find the cut using BFS

4.2 Augmentation

Now, given an augmenting path p, we want to push the maximum permissible flow through it. The edges in the augmenting path p are updated in the similar manner as in the usual max-flow algorithm. In addition to that, for each cross edge $e_{ij:\lambda\mu} \in \hat{\mathcal{E}}_c$ that is in the augmenting path, the exit-flows are updated as follows:

$$\Sigma_{ij:\lambda} = \Sigma_{ij:\lambda} + \alpha \,, \tag{8}$$
$$\Sigma_{ji:\mu} = \Sigma_{ji:\mu} - \alpha \,,$$

where α is the maximum possible flow along the path p.

After the flow augmentation, the lower-graph needs to be updated to maintain the lowest-cross-edge property. Note that the lowest-cross-edge property may be violated due to the following reasons:

1) A new lowest-cross-edge $e_{ij:\lambda\mu}$ is created due to a flow along the edge $e_{ji:\mu\lambda}$.
2) A new lowest-cross-edge $e_{ij:\lambda\mu}$ is created due to a saturating flow along $e_{ij:\lambda\mu'}$ for some $\mu' < \mu$, *i.e.*, the edge $e_{ij:\lambda\mu'}$ disappears from the Ishikawa graph.

Note that, during an augmentation, if a new lowest-cross-edge is created due to a flow in the opposite direction (case-1 above), then the new lowest-cross-edge is known and the lower-graph can be updated directly, *i.e.*, the new lowest-cross-edge can be stored.

On the other hand, if a cross edge becomes saturated (case-2), then we need to run the flow reconstruction algorithm to find the new lowest-cross-edge and update the lower-graph. This can be done in a memory efficient manner, since it only involves one edge $(i, j) \in \mathcal{E}$ at a time.

4.3 Summary

Our polynomial time memory efficient max flow is summarized in Algorithm 1. Let us briefly explain the subroutines below.

    lower-graph: Given the initial Ishikawa edge capacities ϕ^0, this subroutine constructs the lower-graph (with edge capacities $\bar{\phi}$) by retaining the lowest-cross-edges from each node $U_{i:\lambda} \in \hat{\mathcal{V}}$, for each directed edge $(i, j) \in \mathcal{E}^+$ (see Sec. 4.1). If the input to this subroutine is the Ishikawa capacities ϕ_{ij} corresponding to the edge $(i, j) \in \mathcal{E}$, then it retains the lowest-cross-edges $\bar{\phi}_{ij}$.

    shortest_augmenting_path: Given the lower-graph parameters $\bar{\phi}$, this subroutine finds the shortest augmenting p using BFS, as discussed in Section 4.1.

    augment: Given the path p, this subroutine finds the maximum possible flow through the path and updates the lower-graph and the set of exit-flows, as discussed in Section 4.2. In addition, if a new lowest-cross-edge is created due to a flow in the opposite direction (case-1 in Sec. 4.2), then it also updates the lower-graph capacities $\bar{\phi}$.

    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 capacities ϕ_{ij} corresponding to the given edge (i, j). This is accomplished by solving a small max-flow problem (see Sec. 3.1).

    get_labelling: This subroutines finds the partition of the lower-graph by running BFS.

As discussed above, the exit-flows Σ require $\mathcal{O}(\ell)$ storage for each edge $(i, j) \in \mathcal{E}$. In addition, the lower-graph can have at most $\mathcal{O}(|\mathcal{V}|\ell)$ nodes and $\mathcal{O}(|\mathcal{E}|\ell)$ edges. Furthermore, recall that we assume that the initial Ishikawa edge capacities ϕ^0 can be stored efficiently. Therefore, ultimately, the space complexity of our algorithm is $\mathcal{O}((|\mathcal{V}| + |\mathcal{E}|)\ell) = \mathcal{O}(|\mathcal{E}|\ell)$. Let us now prove the polynomial time bound of our algorithm.

4.4 Time Complexity Analysis

We follow the time complexity analysis of the standard Edmonds-Karp algorithm [16] to derive a polynomial time bound on our algorithm. In particular, first the analysis proves that the shortest path distance from source (node 0) to any node is monotonically increasing with each flow augmentation. Then, it derives a bound on the number of augmentations. In fact, the number of augmentations of our MEMF algorithm also has the same bound as the Edmonds-Karp algorithm.

Theorem 4.2. *If the MEMF algorithm is run on the Ishikawa graph* $\hat{\mathcal{G}} = (\hat{\mathcal{V}} \cup \{0, 1\}, \hat{\mathcal{E}})$ *with source* 0 *and terminal* 1, *then the total number of augmentations performed by the algorithm is* $\mathcal{O}(|\hat{\mathcal{V}}||\hat{\mathcal{E}}|)$.

Proof. The proof follows the steps of standard proof of the Edmonds-Karp algorithm. See Appendix A for details. □

Let us analyze the time complexity of each subroutine below. Note that, both the subroutines *shortest_augmenting_path* and *augment* runs in $\mathcal{O}(|\mathcal{E}|\ell)$ time, since, in the worst case, both subroutines need to check each edge in the lower-graph. However, *compute_edges* required to run the flow reconstruction algorithm which takes $\mathcal{O}(\ell^3)$ time for each variable pair (i, j) (assuming a small max-flow problem with 2ℓ nodes and ℓ^2 edges, solved using the most efficient algorithm [17], see Sec. 3.1). Also *lower-graph* requires $\mathcal{O}(\ell^2)$ time for each variable pair (i, j), since it needs to check each of the Ishikawa edges. Hence, the worst case running time of each iteration (*i.e.*, augmentation step) is $\mathcal{O}(|\mathcal{E}|\ell + K(\ell^3 + \ell^2)) = \mathcal{O}(|\mathcal{E}|\ell + |\mathcal{E}|\ell^3) = \mathcal{O}(|\mathcal{E}|\ell^3)$, where K is the maximum number of flow-reconstructions (*i.e.*, saturated cross edges) at an augmentation step. Since the number of augmentations is bounded by $\mathcal{O}(|\hat{\mathcal{V}}||\hat{\mathcal{E}}|)$, the worst case running time of the entire execution of the MEMF algorithm is $\mathcal{O}(|\mathcal{V}|\ell|\mathcal{E}|\ell^2|\mathcal{E}|\ell^3) = \mathcal{O}(|\mathcal{V}||\mathcal{E}|^2\ell^6)$. This is $\mathcal{O}(\ell)$ slower than the standard Edmonds-Karp algorithm on the Ishikawa graph. Note that, however, MEMF requires $\mathcal{O}(\ell)$ less memory.
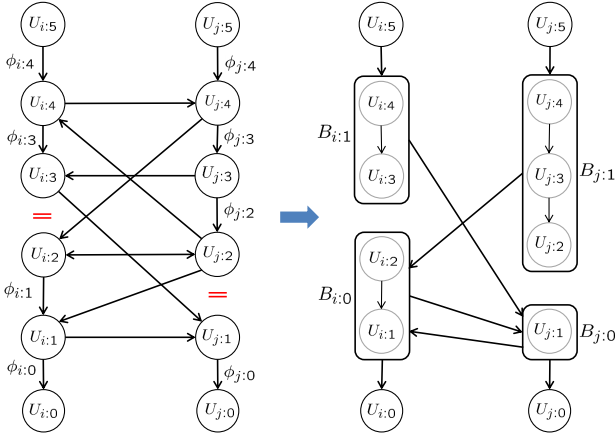
Fig. 4: *To find an augmenting path in a memory efficient manner, we propose a simplified representation of the Ishikawa graph in terms of blocks corresponding to consecutive non-zero edges in each column i.*

5 EFFICIENT ALGORITHM

In the previous section, we have provided a general purpose polynomial time max-flow algorithm that is also memory efficient. However, for computer vision applications, the BK algorithm [7] is shown to be significantly faster than the standard max-flow implementations, even though it lacks the polynomial time guarantee. The basic idea is to maintain a search tree throughout the algorithm instead of building the search tree from scratch at each iteration.

Motivated by this, we also propose doing search-tree-recycling similar to the BK algorithm. Since we lose the polynomial time guarantee, for increased efficiency, we further simplify the Ishikawa graph. In particular, we find an augmenting path in a *block-graph*, that amalgamates the nodes in each column into blocks. Since an augmenting path in our block-graph corresponds to a collection of augmenting paths in the Ishikawa graph, our algorithm converges in fewer iterations than the BK algorithm.

5.1 Efficiently Finding an Augmenting Path

As mentioned above, we find an augmenting path in a block-graph[4], whose construction is detailed below.

Given the parameters 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 $\phi_{i:\lambda}$. 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 block-graph then contains only the blocks and the edges between the blocks.

The edges in the block-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 $\phi_{ij:\lambda\mu} > 0$ 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:

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 $B_{i:\gamma} \to B_{j:\delta}$ 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 block-graph can be characterized by the following theorem.

Theorem 5.1. *Given the set of Ishikawa graph parameters ϕ, there is an augmenting path in the block-graph if and only if there exists an augmenting path in the Ishikawa graph.*

Proof. The basic idea of this proof is the same as the proof of Theorem 4.1. See Appendix B. □

Note that the block-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 $B_{i:\gamma} \to B_{j:\delta}$ is not important, but it is important to have these edges. Note also that the block-graph is constructed incrementally for each edge $(i, j) \in \mathcal{E}$. Hence, it only requires us to store the Ishikawa graph parameters ϕ_{ij} corresponding to the edge (i, j). Furthermore, since the block-graph $\mathcal{G}_b$ is sparse, an augmenting path can be found fairly quickly.

Furthermore, similar to the BK algorithm, we find an augmenting path P_b using BFS and maintain the search tree throughout the algorithm, by repairing it whenever the block-graph is updated. However, since the block-graph needs to be reconstructed after each augmentation, for simplicity, we maintain a single tree[5]. More specifically, we grow the search tree from source (node 0), in a breadth first manner, and if sink (node 1) is reached, then the augmenting path P_b is found.

5.2 Augmentation in the Block-graph

Now, given an augmenting path P_b in the block-graph, we want to push the maximum permissible flow through it. More specifically, since P_b corresponds to a set of augmenting paths $\{p_b\}$ in the Ishikawa graph, we will push the maximum flow through each path p_b, 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_b, and then finding each of the augmenting path p_b 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.

---

4. We called this a *simplified graph* in [13].

5. The BK algorithm maintains two trees, source-tree and sink-tree, but we only maintain the source-tree.
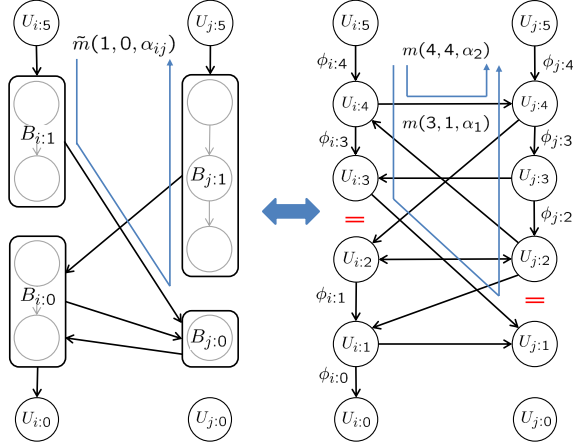
Fig. 5: *An example flow-loop $\tilde{m}(1, 0, \alpha_{ij})$ in the block-graph (left) is equivalent to the summation of two flow-loops $m(3, 1, \alpha_1)$ and $m(4, 4, \alpha_2)$ in the Ishikawa graph (right), with $\alpha_{ij} = \alpha_1 + \alpha_2$.*

Instead, we propose breaking down the augmentation operation in the block-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 5.1. 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

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

Definition 5.2. A flow-loop $\tilde{m}(\gamma, \delta, \alpha)$ in the block-graph $\mathcal{G}_b$ 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_b 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_b, using the following greedy approach.

For each edge $B_{i:\gamma} \rightarrow B_{j:\delta}$ that is part of the path P_b, 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 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
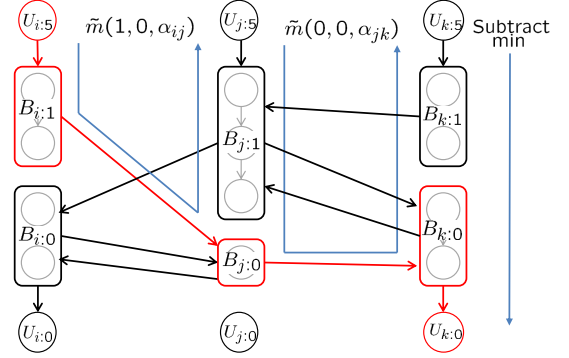

Fig. 6: *An augmentation operation is broken down into a sequence of flow-loops $\tilde{m}(\gamma, \delta, \alpha)$, and a subtraction along the column k. The augmenting path P_s is highlighted in red.*

Algorithm 2 Memory Efficient Max Flow (MEMF) - Efficient Version

---

Require: ϕ^0 ▷ Initial Ishikawa capacities
   $\Sigma \leftarrow 0, T \leftarrow \emptyset$ ▷ Initialize exit-flows and search tree
   $\mathcal{G}_b \leftarrow \text{block-graph}(\phi^0)$ ▷ Initial block-graph
   repeat
     $(T, P_b) \leftarrow \text{augmenting\_path}(\mathcal{G}_b, T)$ ▷ Sec. 5.1
     $\Sigma \leftarrow \text{augment}(P_b, \phi^0, \Sigma)$ ▷ Sec. 5.2
     for each edge $(i, j) \in \mathcal{E}$ affected by augmentation **do**
       $\phi_{ij} \leftarrow \text{compute\_edges}(\phi^0, \Sigma, i, j)$ ▷ Sec. 3.1
       $\mathcal{G}_b^{ij} \leftarrow \text{block-graph}(\phi_{ij}, i, j)$ ▷ Sec. 5.1
     $T \leftarrow \text{repair\_tree}(T, \mathcal{G}_b)$ ▷ Repair search tree
   until no augmenting paths possible
   return get_labelling(T) ▷ Read from search tree

$\mu \geq \check{\mu}$, 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_b.

Since, for each edge (i, j), we do not store all the $2\ell^2$ capacities, but only the 2ℓ 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

$$\Sigma_{ij:3} = \Sigma_{ij:3} + \alpha_1 , \qquad (9)$$
$$\Sigma_{ji:1} = \Sigma_{ji:1} - \alpha_1 ,$$
$$\Sigma_{ij:4} = \Sigma_{ij:4} + \alpha_2 ,$$
$$\Sigma_{ji:4} = \Sigma_{ji:4} - \alpha_2 .$$

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 $\mathcal{G}_b^{ij}$ 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.

5.3 Summary

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

block-graph: Given the initial Ishikawa parameters ϕ^0, this subroutine constructs the block-graph by amalgamating nodes into blocks as described in Section 5.1. 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 block-graph portion $\mathcal{G}_b^{ij}$.

augmenting_path: Given the block-graph $\mathcal{G}_b$ and the search tree T, this subroutine finds an augmenting path P_b by growing the search tree, as discussed in Section 5.1.

augment: Given the path P_b, 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 5.2.

compute_edges: This is the same subroutine as in Algorithm 1. (see Sec. 4.3).

repair_tree: This subroutine is similar to the *adoption* stage of the BK algorithm. Given the reconstructed block-graph, the search tree T is repaired by checking for valid *parent* for each *orphan* node. See section 3.2.3 in [7] for more detail.

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

As discussed Section 4.3, the space complexity of our algorithm is $\mathcal{O}(|\mathcal{E}| \ell)$. For the rest of the paper, this efficient version of the algorithm is referred to as MEMF.

6 EQUIVALENCE WITH MESSAGE PASSING

In this section, we will give a more insightful interpretation of our max-flow algorithm, by showing equivalence with the min-sum message passing algorithm. In particular, first we will characterize the notion of an augmenting path in the message passing context. Then, we will show that, the max-flow algorithm is, in spirit, equivalent to min-sum message passing, for multi-label submodular MRFs. Finally, we observe the relationship between the set of exit-flows in our algorithm and the set of messages in the message passing algorithm. To this end, first let us define the multi-label graph which will be used to explain the equivalence.

6.1 The Mutli-label Graph

An alternative way of representing the multi-label energy function (1) is by defining *indicator variables* $x_{i:\lambda} \in \{0, 1\}$, where $x_{i:\lambda} = 1$ if and only if $x_i = \lambda$. For a given i, exactly one of $x_{i:\lambda}$; $\lambda \in \mathcal{L}$ can have value 1. In terms of the indicator variables, the energy function (1) may be written as

$$E_\theta(\mathbf{x}) = \sum_{i \in \mathcal{V}} \sum_{\lambda \in \mathcal{L}} \theta_{i:\lambda} \, x_{i:\lambda} + \sum_{(i,j) \in \mathcal{E}} \sum_{\lambda, \mu \in \mathcal{L}} \theta_{ij:\lambda\mu} \, x_{i:\lambda} \, x_{j:\mu} \, ,$$

(10)

where the values θ are a particular set of parameters determining the energy function. One may define a graph, called a *multi-label graph*, with nodes denoted by $X_{i:\lambda}$; $i \in \mathcal{V}$, $\lambda \in \mathcal{L}$, as shown in Fig. 7. This graph represents the energy function. Given a labelling $\mathbf{x}$, the value of the energy function is obtained by summing the weights on all nodes
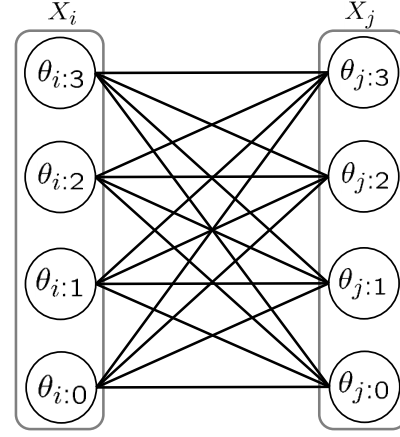

Fig. 7: *Example of a multi-label grpah. Here the nodes represent the unary potentials $\theta_{i:\lambda}$ and the edges represent the pairwise potentials $\theta_{ij:\lambda\mu}$.*

with $x_{i:\lambda} = 1$ (in other words $x_i = \lambda$) plus the weights $\theta_{ij:\lambda\mu}$ such that $x_{i:\lambda} = 1$ and $x_{j:\mu} = 1$.

Furthermore, given the Ishikawa edge capacities ϕ the parameters θ can be calculated using Eq. 4. Note that, in this case, $E_\theta(\mathbf{x}) = E_\phi(\mathbf{x})$ for all labellings $\mathbf{x}$.

6.1.1 Reparametrization

The energy function (10) can be written in different ways as a sum of unary and pairwise terms . In particular, there may exist a different set of parameters θ' such that $E_\theta(\mathbf{x}) = E_{\theta'}(\mathbf{x})$ for all $\mathbf{x}$, denoted as $E_\theta \equiv E_{\theta'}$. The conditions under which $E_\theta \equiv E_{\theta'}$ are well known [10], [15].

Lemma 6.1. *Two energy functions E_θ and $E_{\theta'}$ are equivalent if and only if there exist values $m_{ij:\lambda}$ and $m_{ji:\mu}$ for $(i,j) \in \mathcal{E}$ and $\lambda, \mu \in \mathcal{L}$ such that*

$$\theta'_{ij:\lambda\mu} = \theta_{ij:\lambda\mu} - m_{ij:\lambda} - m_{ji:\mu} \, ,$$
$$\theta'_{i:\lambda} = \theta_{i:\lambda} + \sum_{(k,i) \in \mathcal{E}^+} m_{ik:\lambda} \, .$$

The values of $m_{ij:\lambda}$ constitute a *message m_{ij} passed from* the edge (i, j) to the node i; it may be thought of as a message vector (indexed by λ). A message m_{ij} causes values $m_{ij:\lambda}$ to be *swept out* of all the edges $\theta_{ij:\lambda\mu}$ and adds onto the nodes $\theta_{i:\lambda}$. Messages are passed in both directions from an edge (i, j).

Note that, reparametrization provides an alternative way of implementing the min-sum message passing algorithm. In particular, the objective of min-sum message passing is to reparametrize the energy function so that the constant term θ_c is maximized, while keeping the parameters θ non-negative, where the constant term is defined as follows:

$$\theta_c = \sum_{i \in \mathcal{V}} \min_{\lambda \in \mathcal{L}} \theta_{i:\lambda} \, .$$

(11)

For more details see [10], [18].

6.2 The Notion of an Augmenting Path in the Multi-label Graph

We will now give an equivalent notion of an augmenting path in the context of the multi-label graph. To this end,

we will first understand the motivation behind finding an augmenting path in the Ishikawa graph.

Let us consider an augmenting path in the Ishikawa graph. If it is a *trivial* augmenting path, *i.e.*, it is an augmenting path along a column from nodes $U_{i:\ell}$ to $U_{i:0}$, then pushing the maximum flow along the path translates into subtracting the minimum value from each $\phi_{i:\lambda}$; $\lambda \in \mathcal{L}$. In the multi-label graph, it is trivially equivalent to subtracting the minimum from each $\theta_{i:\lambda}$; $\lambda \in \mathcal{L}$.

Let us consider a more interesting augmenting path in the Ishikawa graph, which contains at least one cross edge $e_{ij:\lambda\mu} \in \hat{\mathcal{E}}_c$. Similar to the discussion in Section 5.2, one can easily see that, the augmentation operation can be broken down into a sequence of flow-loops $m(\lambda, \mu, \alpha)$ (see Def. 5.1) and a subtraction along a column. This intuitively suggests that an augmenting path in the Ishikawa graph can be translated into a trivial augmenting path by passing flow around loops, *i.e.*, they differ by a *null* flow. Therefore, the motivation of finding an augmenting path, is to pass flow around loops to get a trivial augmenting path.

Note that, the notion of a trivial augmenting path in the multi-label graph is,

$$\theta_{i:\lambda} > 0 \quad \forall \lambda \in \mathcal{L} , \tag{12}$$

for some $i \in \mathcal{V}$. Furthermore, by Lemma 3.1 and by Lemma 6.1, a flow-loop (or null flow) corresponds to a reparametrization of the multi-label energy function. Hence, the notion of an augmenting path in the multi-label graph can be characterized as, finding a set of reparametrizations that makes $\theta_{i:\lambda}$ positive for all $\lambda \in \mathcal{L}$ for some $i \in \mathcal{V}$.

6.3 Equivalence of Max-flow and Min-sum Message Passing

Note that, as we have discussed above, an augmenting path in the Ishikawa graph can be translated into a trivial augmenting path by passing flow around loops. Furthermore, pushing the maximum flow through a trivial augmenting path is simply a subtraction of the minimum value $\min_{\lambda \in \mathcal{L}} \phi_{i:\lambda}$ from each $\phi_{i:\lambda}$. In fact, one can accumulate the total flow passed from source to sink, which is exactly the constant term defined in Eq. 11. Hence, max-flow tries to maximize θ_c, by passing flow around loops (*i.e.*, reparametrizing), while keeping the edge capacities ϕ non-negative. This is, in spirit, equivalent to the min-sum message passing algorithm. Note that, the optimality of min-sum message passing for the case of multi-label submodular MRFs, is observed in [15], [19].

6.4 Flow-loop as a Reparametrization

As mentioned above, from Lemma 3.1 and Lemma 6.1, it is clear that, a flow-loop corresponds to a reparametrization of the multi-label energy function. In this section, we will find the equivalent reparametrization of a flow-loop $m(\lambda, \mu, \alpha)$. This will later allow us to understand the relationship between the set of exit-flows and the set of messages. Let us now state and prove our theorem.
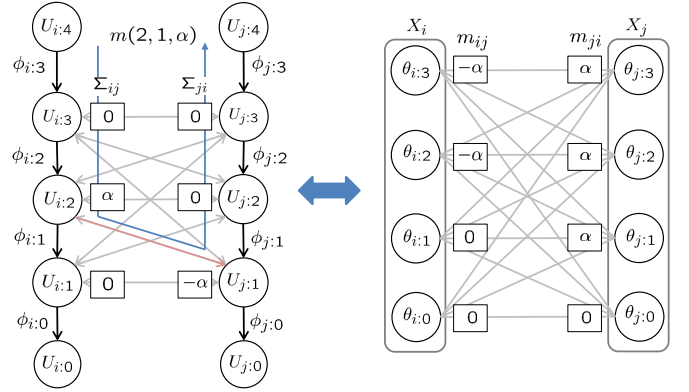

Fig. 8: *A flow $m(2, 1, \alpha)$ in the Ishikawa graph (left) and its equivalent reparametrization in the multi-label graph (right). Note that, the exit-flow vectors $(\Sigma_{ij}, \Sigma_{ji})$ and the corresponding message vectors (m_{ij}, m_{ji}) are shown next to the nodes.*

Theorem 6.1. *Applying a flow-loop $m(\lambda, \mu, \alpha)$ in the Ishikawa graph is equivalent to a reparametrization in the multi-label graph, with messages*

$$m_{ij:\lambda'} = -\alpha \quad \forall \lambda' \geq \lambda , \tag{13}$$
$$m_{ji:\mu'} = \alpha \quad \forall \mu' \geq \mu .$$

Proof. Let the Ishikawa parameters be ϕ and the multi-label graph parameters be θ and assume that the flow is applied between columns i and j. Also after the flow the parameters be ϕ' and θ' respectively. Since θ can be calculated from ϕ using Eq. 4, $E_\theta \equiv E_\phi$. Similarly $E_{\phi'} \equiv E_{\theta'}$. Also from Lemma 3.1, $E_\phi \equiv E_{\phi'}$. Hence, $E_\theta \equiv E_\phi \equiv E_{\phi'} \equiv E_{\theta'}$. Now from Definition 5.1,

$$\phi'_{i:\lambda'} = \phi_{i:\lambda'} - \alpha \quad \forall \lambda' \geq \lambda , \tag{14}$$
$$\phi'_{j:\mu'} = \phi_{j:\mu'} + \alpha \quad \forall \mu' \geq \mu ,$$
$$\phi'_{ij:\lambda\mu} = \phi_{ij:\lambda\mu} - \alpha ,$$
$$\phi'_{ji:\mu\lambda} = \phi_{ji:\mu\lambda} + \alpha .$$

Substituting in Eq. 4,

$$\theta'_{i:\lambda'} = \theta_{i:\lambda'} - \alpha \quad \forall \lambda' \geq \lambda , \tag{15}$$
$$\theta'_{j:\mu'} = \theta_{j:\mu'} + \alpha \quad \forall \mu' \geq \mu ,$$
$$\theta'_{ij:\lambda'\mu'} = \theta_{ij:\lambda'\mu'} - \alpha \quad \forall \lambda' < \lambda, \ \mu' \geq \mu ,$$
$$\theta'_{ij:\lambda'\mu'} = \theta_{ij:\lambda'\mu'} + \alpha \quad \forall \lambda' \geq \lambda, \ \mu' < \mu .$$

Now, since $E_\theta \equiv E_{\theta'}$, by Lemma 6.1, there exists messages $m_{ij:\lambda}$ and $m_{ji:\mu}$ such that,

$$\theta'_{ij:\lambda\mu} = \theta_{ij:\lambda\mu} - m_{ij:\lambda} - m_{ji:\mu} , \tag{16}$$
$$\theta'_{i:\lambda} = \theta_{i:\lambda} + \sum_{(k,i) \in \mathcal{E}^+} m_{ki:\lambda} .$$

With a little bit of calculation, one can see that, the messages take the following form

$$m_{ij:\lambda'} = -\alpha \quad \forall \lambda' \geq \lambda , \tag{17}$$
$$m_{ji:\mu'} = \alpha \quad \forall \mu' \geq \mu .$$

Note that, for a permissible flow $m(\lambda, \mu, \alpha)$, the parameters ϕ' and θ' are non-negative. $\qquad\square$

This equivalence is shown in Fig. 8 for an example flow-loop $m(2, 1, \alpha)$. Note that, as shown in the figure, the

flow α through an edge $\phi_{ij:\lambda\mu}$ may be recorded in the set of exit-flows Σ. Furthermore, as shown in the figure, the relationship between the set of exit-flows and the set of messages can be written as,

$$\Sigma_{ij:\lambda} = m_{ij:\lambda-1} - m_{ij:\lambda} \quad \forall \lambda \in \{1, \cdots, \ell-1\} , \quad (18)$$
$$\Sigma_{ji:\mu} = m_{ji:\mu-1} - m_{ji:\mu} \quad \forall \mu \in \{1, \cdots, \ell-1\} .$$

7 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.

7.1 Max-flow-based Methods

The most popular method to minimize a multi-label submodular MRF energy is to construct the Ishikawa graph [1] 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 [3], the push-relabel approach [20] and the pseudo-flow techniques [21]. Even though numerous implementations are available, the BK method [7] is arguably the fastest implementation for 2D and sparse 3D graphs. Recently, for dense problems, the IBFS algorithm [22] was shown to outperform the BK method in a number of experiments [23]. 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 [24], [25]. The second one makes use of distributed computing [9], [26], [27]. 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 [9], it can also be parallelized.

7.2 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 [18]. Many such algorithms have been studied [10], [11], [12], [15]. Even though these algorithms are good at approximating the optimal solution (also theoretically optimal for multi-label submodular MRFs [19]), as evidenced by the comparison of [28] and by our experiments, they usually take much longer to converge to the optimal solution than max-flow-based techniques.

8 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 [29], [30]: Tsukuba, Venus, Sawtooth, Map, Cones and Teddy, and one instance from the KITTI dataset [31] (see Fig. 9). For Tsukuba and Venus, we used the unary potentials of [32], and for all other stereo

Fig. 9: *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.*
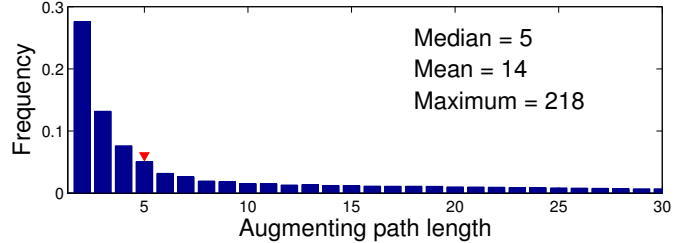

Fig. 10: *Lengths of augmenting paths found by our algorithm for the Tsukuba stereo instance (see Sec. 8.1). Each bar indicates the proportion of paths of a certain length. For example, out of all augmenting paths 28% of them were of length 2. The red arrow indicates the median length.*

cases, those of [33]. For inpainting, we used the Penguin and House images employed in [32], and we used the same unary potentials as in [32]. In all the above cases, we used pairwise potentials that can be expressed as

$$\theta_{ij}(x_i, x_j) = w_{ij}\, \theta(|x_i - x_j|) , \quad (19)$$

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

We compare our results with two max-flow implementations: the BK algorithm [7] and Excesses Incremental Breadth First Search (EIBFS) [34] (which we ran on the Ishikawa graph), and three LP relaxation-based algorithms: Tree Reweighted Message Passing (TRWS) [10], Subgradient based Dual Decomposition (DDSG) [11] and the Adaptive Diminishing Smoothing algorithm (ADSal) [12]. For DDSG and ADSal, we used the Opengm [35] implementations. For the other algorithms, we employed the respective authors' implementations.

In practice, we only ran the BK algorithm 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 algorithm and EIBFS, and is an order of magnitude faster than state-of-the-art message-passing algorithms.

8.1 MEMF analysis

In this section, we empirically analyze various properties of our algorithm. First, note that, at each iteration, *i.e.,*

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 for quadratic regularizer (see para. 2 of Sec. 8, for details on the algorithms). 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.*

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. 10, 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 the block-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 algorithm.

Next, we fixed the number of labels but varied the image size and compare the running times of the max-flow algorithms, for Tsukuba and Penguin instances in Fig. 11a, 11b. Similarly, we fixed the image size but varied the number of labels and report the running times in Fig. 11c, 11d. By doing this, we try to estimate the empirical time complexity of our algorithm. Note that, similar to other max-flow algorithms, MEMF exhibited near-linear performance with respect to the image size and near-cubic performance with respect to the number of labels, in these experiments.

Finally, we report the percentage of time taken by each subroutine of our algorithm, for Tsukuba and Penguin instances in Fig. 12. Note that the individual time complexities of the subroutines *compute_edges* and *block-graph* are $\mathcal{O}(\ell^3)$ and $\mathcal{O}(\ell^2)$, respectively. Therefore, they become dominant when the number of labels is large, and hence the corresponding percentages of time are high, particularly for Penguin.

8.2 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 [6], which minimizes MRFs with some non-convex pairwise potentials (or regularizers) by iteratively building and solving an Ishikawa graph. This lets us tackle much larger non-submodular problems. In particular, we computed inpaint-
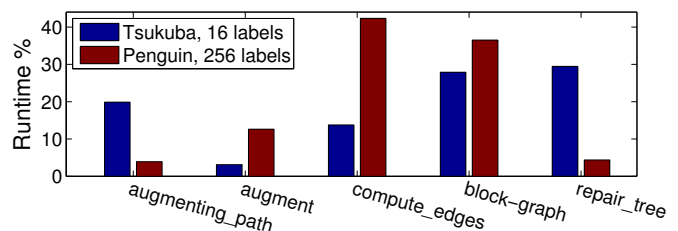

Fig. 12: *Percentage of time taken by each subroutine (see Sec. 8.1). Note that, in Penguin, due to large number of labels, the percentages of time spend on compute_edges and block-graph are high.*

Problem	Memory [MB]		Time [s]	
	BK	MEMF	BK	MEMF
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 (see Sec. 8.2). Here, "Penguin-128/10" corresponds to the Penguin problem with 128 labels and the truncated quadratic function with truncation value 10 as regularizer. A "*" indicates a memory estimate. Compared to BK method, MEMF is only $4 - 11$ times slower but requires $13 - 18$ times less memory, which makes it applicable to much larger MRFs.*

ing results on Penguin by using all 256 labels, as opposed to the down-sampled label sets used in [6]. The results of the IRGC+expansion algorithm, with the BK method and with MEMF are summarized in Table 2.

8.3 Robust Regularizer

Since robust regularizers are highly effective in computer vision, we tested our algorithm by choosing Huber loss function [36] as the regularizer,

$$\theta(|x_i - x_j|) = \begin{cases} \frac{1}{2}|x_i - x_j|^2 & \text{if } |x_i - x_j| \leq \delta \\ \delta\left(|x_i - x_j| - \frac{1}{2}\delta\right) & \text{otherwise ,} \end{cases} \quad (20)$$

where δ is the Huber value. The results are summarized in Table 3. In this experiment, the Huber value was set to 4 for Tsukuba, Venus and Sawtooth, 6 for Map, 20 for Cones and Teddy, 10 for KITTI and 25 for Penguin and House. Note that, the Ishikawa graph for a Huber regularizer is significantly smaller, *i.e.*, the number of edges per variable
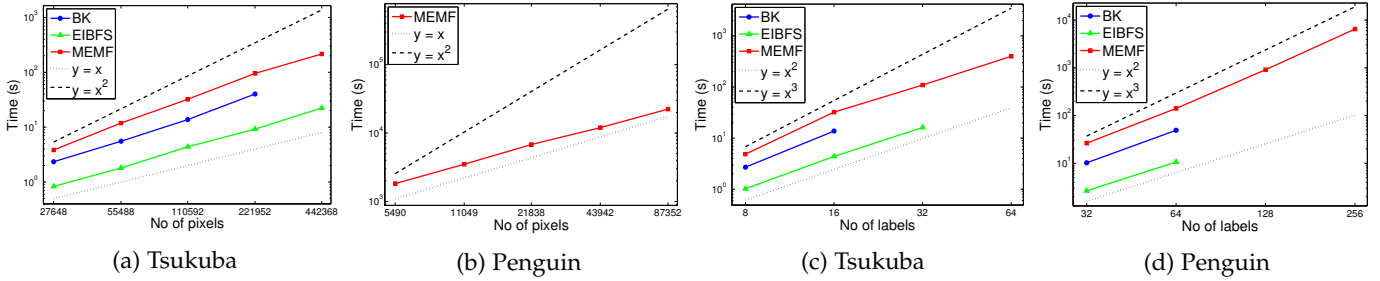
(a) Tsukuba   (b) Penguin   (c) Tsukuba   (d) Penguin

Fig. 11: *Running time plots (in logarithmic scale) by changing the image size (**a**, **b**) and by changing the number of labels (**c**, **d**), for Tsukuba and Penguin (see Sec. 8.1). The dashed lines provide the reference slopes. Note that, all algorithms exhibited near-linear performance with respect to the number of pixels and near-cubic performance with respect to the number of labels, but MEMF required $\mathcal{O}(\ell)$ less memory. The plots of BK algorithm and EIBFS are not complete, since we could not run them due to excessive memory requirement.*
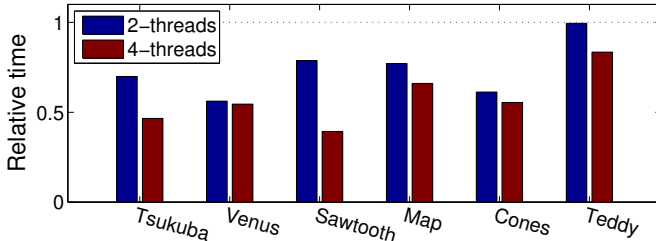

Fig. 13: *Our algorithm can be accelerated using the parallel max-flow technique (see Sec. 8.4). The relative times ranged from 0.56 to 0.99 with 2-threads and from 0.39 to 0.83 with 4-threads. In Teddy, in the case of 2-threads, the multi-threaded algorithm performs almost the same as the single-threaded algorithm, which may be due to bad splits.*

pair is $\mathcal{O}(\delta\ell)$, instead of $\mathcal{O}(\ell^2)$. Even in this case, our algorithm lets us solve much larger problems than the BK algorithm and EIBFS, and is an order of magnitude faster than state-of-the-art message-passing algorithms.

8.4 Parallelization

We parallelized our algorithm based on the dual-decomposition technique of [9] and evaluated on the same six stereo instances from the Middlebury dataset [29], [30]. The relative times t_m/t_s, where t_m stands for the multi-thread time and t_s for the single-thread one, are shown in Fig. 13 for two and four threads. In this experiment, for all problems, the image grid was split vertically into two and four equally-sized blocks, respectively. Note that this spliting strategy is fairly arbitrary, and may affect the performance of the multi-threaded algorithm. In fact finding better splits may itself be a possible future direction.

9 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.

Appendix A
Time Complexity Analysis of the Polynomial time version of MEMF

Let us denote the Ishikawa graph with $\hat{\mathcal{G}} = (\hat{\mathcal{V}} \cup \{0, 1\}, \hat{\mathcal{E}})$ and the lower-graph with $\hat{\mathcal{G}}^s = (\hat{\mathcal{V}} \cup \{0, 1\}, \hat{\mathcal{E}}^s)^6$. In this section, we denote the nodes with u, v, etc. Also the notation $u_1 \leq u$ means, the node u_1 and u are in the same column where u_1 is below u. Let $\hat{\mathcal{G}}^s_f$ denotes the residual graph of the lower-graph after the flow f and similarly $\hat{\mathcal{E}}^s_f$ denotes the set of non-zero residual edges. Let $d_f(u, v)$ denotes the shortest path distance from u to v calculated by MEMF.

Lemma A.1. *If the MEMF algorithm is run on the Ishikawa graph $\hat{\mathcal{G}} = (\hat{\mathcal{V}} \cup \{0, 1\}, \hat{\mathcal{E}})$ with source 0 and terminal 1, then for any node $v \in \hat{\mathcal{V}}$, the shortest path distance $d_f(0, v)$ in the residual lower-graph $\hat{\mathcal{G}}^s_f$ increases monotonically with each flow augmentation.*

Proof. We will suppose that for some node $v \in \hat{\mathcal{V}}$, there is a flow augmentation that causes the shortest path distance from 0 to v to decrease, and then we will derive a contradiction. Let f be the flow just before the first augmentation that decreases some shortest path distance, and let f' be the flow just afterward. Let v be the node with the minimum $d_{f'}(0, v)$ whose distance was decreased by the augmentation, so that $d_{f'}(0, v) < d_f(0, v)$. Let $p = 0 \rightsquigarrow u \to v$ be a shortest path from 0 to v in $\hat{\mathcal{G}}^s_{f'}$, so that $(u, v) \in \hat{\mathcal{E}}^s_{f'}$ and

$$d_{f'}(0, v) = \begin{cases} d_{f'}(0, u) & \text{if } u < v \text{ (infinite edge)} \\ d_{f'}(0, u) + 1 & \text{otherwise} . \end{cases} \tag{21}$$

Because of how we chose v, we know that the distance of node u from the source 0 did not decrease, *i.e.*,

$$d_{f'}(0, u) \geq d_f(0, u) . \tag{22}$$

We claim that $(u, v) \notin \hat{\mathcal{E}}^s_f$. Why? If we had $(u, v) \in \hat{\mathcal{E}}^s_f$, then we would also have

$$\begin{aligned} d_f(0, v) &\leq d_f(0, u) + 1 , \\ &\leq d_{f'}(0, u) + 1 , \\ &= d_{f'}(0, v) , \end{aligned} \tag{23}$$

---

6. The superscript s is used to restate the fact, that the lower-graph is a subgraph of the Ishikawa graph.

Problem	Memory [MB]				Time [s]			
	BK	EIBFS	TRWS	MEMF	BK	EIBFS	TRWS	MEMF
Tsukuba	1715	1385	287	**211**	8	**3**	198	28
Venus	3375	2719	638	**396**	17	**5**	211	57
Sawtooth	3348	2698	633	**393**	15	**4**	467	34
Map	2680	2116	494	**219**	22	**5**	>2953	36
Cones	*42155	*32167	5025	**1200**	-	-	1118	**363**
Teddy	*42155	*32167	5025	**1200**	-	-	6879	**2064**
KITTI	*42161	*32627	6416	**2215**	-	-	>30165	**18923**
Penguin	*33487	*25423	**215**	663	-	-	>50000	**6277**
House	*100494	*76295	**643**	1986	-	-	>50000	**8568**

TABLE 3: *Memory consumption and runtime comparison with state-of-the-art baselines for Huber regularizer (see Sec. 8.3). 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 much lower memory consumption than the max-flow-based methods and is an order of magnitude faster than message-passing algorithms. Compared to EIBFS, our algorithm is $7 - 11$ times slower, but requires $7 - 10$ 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.*

which contradicts our assumption that $d_{f'}(0,v) < d_f(0,v)$. The above argument simply follows even if (u,v) is an infinite capacity edge. Hence $(u,v) \notin \hat{\mathcal{E}}_f^s$.

How can we have $(u,v) \notin \hat{\mathcal{E}}_f^s$ and $(u,v) \in \hat{\mathcal{E}}_{f'}^s$? Note that, in this case, (u,v) cannot be an infinite capacity edge. There can be two reasons:

1) A new lowest edge (u,v) is created due to the flow from v to u. That means the augmentation must have increased the flow from v to u. The MEMF algorithm always augments flow along shortest paths, and therefore the shortest path from 0 to u in $\hat{\mathcal{G}}_f^s$ has (v,u) as its last edge. Therefore,

$$d_f(0,v) = d_f(0,u) - 1 , \qquad (24)$$
$$\leq d_{f'}(0,u) - 1 ,$$
$$= d_{f'}(0,v) - 2 ,$$

which contradicts our assumption that $d_{f'}(0,v) < d_f(0,v)$.

2) A new edge (u,v) is created due to a saturating flow from u to v_1 for some $v_1 < v$. The MEMF algorithm always augments flow along shortest paths, and therefore the shortest path from 0 to v_1 in $\hat{\mathcal{G}}_f^s$ has (u,v_1) as its last edge. Since $d_f(0,v) \leq d_f(0,v_1)$, due to the upward infinite capacity edges, we have,

$$d_f(0,v) \leq d_f(0,v_1) , \qquad (25)$$
$$= d_f(0,u) + 1 ,$$
$$\leq d_{f'}(0,u) + 1 ,$$
$$= d_{f'}(0,v) ,$$

which contradicts our assumption that $d_{f'}(0,v) < d_f(0,v)$.

We conclude that our assumption that such a node v exists is incorrect. □

The next theorem bounds the number of iterations of the MEMF algorithm.

Theorem A.1. *If the MEMF algorithm is run on the Ishikawa graph $\hat{\mathcal{G}} = (\hat{\mathcal{V}} \cup \{0,1\}, \hat{\mathcal{E}})$ with source 0 and sink 1, then the total number of augmentations performed by the algorithm is $\mathcal{O}(|\hat{\mathcal{V}}||\hat{\mathcal{E}}|)$.*

Proof. We say that an edge (u,v) in a residual lower-graph $\hat{\mathcal{G}}_f^s$ is *critical* on an augmenting path p if the residual capacity of p is the residual capacity of (u,v), i.e., if $c_f(p) = c_f(u,v)$.

After we have augmented flow along an augmenting path, any critical edge on the path disappears from the residual graph. Moreover, at least one edge on any augmenting path must be critical. We will show that each of the $|\hat{\mathcal{E}}|$ edges can become critical at most $|\hat{\mathcal{V}}|/2 + 1$ times. Furthermore, note that, an infinite capacity edge cannot be critical at any point of the algorithm.

Let u and v be nodes in $\hat{\mathcal{V}} \cup \{0,1\}$ that are connected by an edge in $\hat{\mathcal{E}}^s$. Since augmenting paths are shortest paths, when (u,v) is critical for the first time, we have

$$d_f(0,v) = d_f(0,u) + 1 . \qquad (26)$$

Once the flow is augmented, the edge (u,v) disappears from the residual graph. Since we maintain the lowest-cross-edge property, there cannot be an edge (u,v_1) in $\hat{\mathcal{G}}_f^s$ for some $v_1 < v$. Therefore, the edge (u,v) cannot reappear later on another augmenting path until after the flow from u to v_1 for some $v_1 \leq v$ is decreased, which occurs only if (v_1,u) appears on an augmenting path. If f' is the flow when this event occurs, then we have

$$d_{f'}(0,u) = d_{f'}(0,v_1) + 1 . \qquad (27)$$

Since $d_{f'}(0,v) \leq d_{f'}(0,v_1)$, due to the upward infinite capacity edges, and $d_f(0,v) \leq d_{f'}(0,v)$ by Lemma A.1, we have

$$d_{f'}(0,u) = d_{f'}(0,v_1) + 1 , \qquad (28)$$
$$\geq d_{f'}(0,v) + 1 ,$$
$$\geq d_f(0,v) + 1 ,$$
$$= d_f(0,u) + 2 .$$

Consequently, from the time (u,v) becomes critical to the time when it next becomes critical, the distance of u from the source increases by at least 2. The distance of u from the source is initially at least 0. The intermediate nodes on a shortest path from 0 to u cannot contain 0, u or 1 (since (u,v) on an augmenting path implies that $u \neq 1$). Therefore, until u becomes unreachable from the source, if ever, its distance is at most $|\hat{\mathcal{V}}|$. Thus, after the first time that (u,v) becomes critical, it can become critical at most $|\hat{\mathcal{V}}|/2$ times more, for a total of $|\hat{\mathcal{V}}|/2 + 1$ times. Since there are $\mathcal{O}(|\hat{\mathcal{E}}|)$ pairs of nodes that can have an edge between them in a residual graph, the total number of critical edges during the entire execution of the MEMF algorithm is $\mathcal{O}(|\hat{\mathcal{V}}||\hat{\mathcal{E}}|)$. Each augmenting path has at least one critical edge, and hence the theorem follows. □

APPENDIX B
PROOF OF THEOREM 5.1

Theorem. *Given the set of Ishikawa graph parameters ϕ, there is an augmenting path in the block-graph if and only if there exists an augmenting path in the Ishikawa graph.*

Proof. First, we will prove that, if there is an augmenting path in the block-graph, then there exists an augmenting path in the Ishikawa graph. It is clear that an augmenting path in the block-graph contains an edge from node 0 to a block and then a sequence of edges $B_{i:\gamma} \to B_{j:\delta}$ and finally an edge from a block to node 1. Note that an edge from node 0 to a block $B_{i:\gamma}$ corresponds to a positive edge $e_{i:\ell-1}$ in the Ishikawa graph; similarly an edge from a block $B_{j:\delta}$ to node 1 corresponds to a positive edge $e_{j:0}$. Now, consider an edge $B_{i:\gamma} \to B_{j:\delta}$ in the augmenting path. Corresponding to this, there exists a positive edge $e_{ij:\lambda\mu}$ such that $U_{i:\lambda} \in B_{i:\gamma'}$ for some $\gamma' \geq \gamma$ and $U_{j:\mu} \in B_{j:\delta}$ in the Ishikawa graph. Also along the column i, there are upward infinite capacity edges, and nodes corresponding to a block are also connected with positive bidirectional edges. Hence, there exists an augmenting path in the Ishikawa graph, corresponding to the augmenting path in the block-graph.

Now, we will prove the converse. Consider an augmenting path in the Ishikawa graph. The path may contain a sequence of positive edges $e_{i:\lambda}$, $e_{ij:\lambda\mu}$ and infinite capacity edges $e_{ii:\lambda\lambda+1}$. Note that, by construction, the $e_{i:\lambda}$ edges either will be in the same block $B_{i:\gamma}$ in the block-graph, or will be between a block and node 0 or node 1. Furthermore, the infinite capacity edges either will be in the same block, or there will be an edge $B_{i:\gamma} \to B_{j:\delta}$ in the block-graph to represent them. Finally, if $e_{ij:\lambda\mu}$ is a positive edge, then, by construction of the block-graph, there exists an edge $B_{i:\gamma} \to B_{j:\delta'}$ where $U_{i:\lambda} \in B_{i:\gamma}$ and $U_{j:\mu} \in B_{j:\delta}$ with $\delta' \leq \delta$. Hence, if there is an augmenting path in the Ishikawa graph, then there exists an augmenting path in the block-graph. $\square$

REFERENCES

[1] H. Ishikawa, "Exact optimization for markov random fields with convex priors," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 25, no. 10, pp. 1333–1336, 2003.

[2] D. Schlesinger and B. Flach, *Transforming an arbitrary minsum problem into a binary one*. TU, Fak. Informatik, 2006.

[3] L. Ford and D. R. Fulkerson, *Flows in networks*. Princeton Princeton University Press, 1962, vol. 1962.

[4] P. Torr and M. Kumar, "Improved moves for truncated convex models," in *Advances in neural information processing systems*, 2009, pp. 889–896.

[5] O. Veksler, "Multi-label moves for mrfs with truncated convex priors," *International journal of computer vision*, vol. 98, no. 1, pp. 1–14, 2012.

[6] 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.

[7] 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*, vol. 26, no. 9, pp. 1124–1137, 2004.

[8] 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*, vol. 2. IEEE, 2005, pp. 922–929.

[9] P. Strandmark and F. Kahl, "Parallel and distributed graph cuts by dual decomposition," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 2085–2092.

[10] V. Kolmogorov, "Convergent tree-reweighted message passing for energy minimization," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 28, no. 10, pp. 1568–1583, 2006.

[11] N. Komodakis, N. Paragios, and G. Tziritas, "Mrf energy minimization and beyond via dual decomposition," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 33, no. 3, pp. 531–552, 2011.

[12] B. Savchynskyy, S. Schmidt, J. H. Kappes, and C. Schnörr, "Efficient mrf energy minimization via adaptive diminishing smoothing," in *Uncertainty in Artificial Intelligence*, 2012, pp. 746–755.

[13] T. Ajanthan, R. Hartley, and M. Salzmann, "Memory efficient max flow for multi-label submodular mrfs," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[14] E. Boros and P. L. Hammer, "Pseudo-boolean optimization," *Discrete applied mathematics*, vol. 123, no. 1, pp. 155–225, 2002.

[15] T. Werner, "A linear programming approach to max-sum problem: A review," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, no. 7, pp. 1165–1179, 2007.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press Cambridge, 2001, vol. 6.

[17] J. B. Orlin, "Max flows in o (nm) time, or better," in *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. ACM, 2013, pp. 765–774.

[18] 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*, vol. 51, no. 11, pp. 3697–3717, 2005.

[19] V. Kolmogorov and M. Wainwright, "On the optimality of tree-reweighted max-product message-passing," *Uncertainty on Artificial Intelligence*, 2005.

[20] A. V. Goldberg and R. E. Tarjan, "A new approach to the maximum-flow problem," *Journal of the ACM (JACM)*, vol. 35, no. 4, pp. 921–940, 1988.

[21] B. G. Chandran and D. S. Hochbaum, "A computational study of the pseudoflow and push-relabel algorithms for the maximum flow problem," *Operations research*, vol. 57, no. 2, pp. 358–376, 2009.

[22] 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*. Springer, 2011, pp. 457–468.

[23] T. Verma and D. Batra, "Maxflow revisited: An empirical comparison of maxflow algorithms for dense vision problems." in *BMVC*, 2012, pp. 1–12.

[24] 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*. IEEE, 2008, pp. 1–8.

[25] O. Jamriška, D. Sýkora, and A. Hornung, "Cache-efficient graph cuts on structured grids," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 3673–3680.

[26] A. Shekhovtsov and V. Hlaváč, "A distributed mincut/maxflow algorithm combining path augmentation and push-relabel," *International journal of computer vision*, vol. 104, no. 3, pp. 315–342, 2013.

[27] 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*. IEEE, 2008, pp. 1–8.

[28] 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*, pp. 1–30, 2015. [Online]. Available: http://dx.doi.org/10.1007/s11263-015-0809-x

[29] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International journal of computer vision*, vol. 47, no. 1-3, pp. 7–42, 2002.

[30] ——, "High-accuracy stereo depth maps using structured light," in *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, vol. 1. IEEE, 2003, pp. I–195.

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

[32] 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*, vol. 30, no. 6, pp. 1068–1080, 2008.

[33] S. Birchfield and C. Tomasi, "A pixel dissimilarity measure that is insensitive to image sampling," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 20, no. 4, pp. 401–406, 1998.

[34] 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*. Springer, 2015, pp. 619–630.

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

[36] P. J. Huber, "Robust estimation of a location parameter," *The Annals of Mathematical Statistics*, vol. 35, no. 1, pp. 73–101, 1964.

Thalaiyasingam Ajanthan obtained his BSc degree in Electronics and Telecommunication Engineering from University of Moratuwa, Sri Lanka, in 2013. He is now a PhD student at the College of Engineering and Computer Science, Australian National University (ANU). He is also a member of the Analytics Group at Data61, CSIRO. He is a student member of IEEE.

Richard Hartley is a member of the Computer Vision Group in the Research School of Engineering, at ANU, where he has been since January, 2001. He is also a member of the Analytics group in Data61, CSIRO. He worked at the GE Research and Development Center from 1985 to 2001, working first in VLSI design, and later in computer vision. He became involved with Image Understanding and Scene Reconstruction working with GEs Simulation and Control Systems Division. He is an author (with A. Zisserman) of the book Multiple View Geometry in Computer Vision.

Mathieu Salzmann obtained his MSc and PhD degrees from EPFL in 2004 and 2009, respectively. He then joined the International Computer Science Institute and the EECS Department at the University of California at Berkeley as a postdoctoral fellow, later the Toyota Technical Institute at Chicago as a research assistant professor and a senior researcher at NICTA in Canberra. He is now a senior researcher in Computer Vision Lab at EPFL.