Computer Vision and Image Understanding 143 (2016) 80–91

Contents lists available at ScienceDirect

Computer Vision and Image Understanding

journal homepage: www.elsevier.com/locate/cviu

Lazy Generic Cuts

Dinesh Khandelwal [a,*], Kush Bhatia [a], Chetan Arora [b], Parag Singla [a]

[a] Indian Institute of Technology Delhi (IIT Delhi), New Delhi, India
[b] Indraprastha Institute of Information Technology Delhi (IIIT Delhi), New Delhi, India

ARTICLE INFO

Article history:
Received 17 October 2014
Accepted 20 October 2015

Keywords:
Markov random fields
Higher order potential
Graphical models
MRF-MAP
Optimal inference
Generic cuts

ABSTRACT

LP relaxation based message passing and flow-based algorithms are two of the popular techniques for performing MAP inference in graphical models. Generic Cuts (GC) (Arora et al., 2015) combines the two approaches to generalize the traditional max-flow min-cut based algorithms for binary models with higher order clique potentials. The algorithm has been shown to be significantly faster than the state of the art algorithms. The time and memory complexities of Generic Cuts are linear in the number of constraints, which in turn is exponential in the clique size. This limits the applicability of the approach to small cliques only. In this paper, we propose a lazy version of Generic Cuts exploiting the property that in most of such inference problems a large fraction of the constraints are never used during the course of minimization. We start with a small set of constraints (called the *active* constraints) which are expected to play a role during the minimization process. GC is then run with this reduced set allowing it to be efficient in time and memory. The set of active constraints is adaptively learnt over multiple iterations while guaranteeing convergence to the optimum for submodular clique potentials. Our experiments show that the number of constraints required by the algorithm is typically less than 3% of the total number of constraints. Experiments on computer vision datasets show that our approach can significantly outperform the state of the art both in terms of time and memory and is scalable to clique sizes that could not be handled by existing approaches.

1. Introduction

MAP inference in graphical models corresponds to finding the most likely joint assignment to the underlying variables. It is an important problem for a wide variety of applications including natural language processing, computer vision and biology. The problem is known to be NP-hard (in the number of variables) except for some special cases, such as when the clique potentials are submodular [1–4]. Irrespective of submodularity, the complexity of the algorithms suggested in computer vision has been exponential in the size of the clique. This makes inference intractable even for mid-sized cliques.

Two of the popular approaches for solving the MAP problem include: LP relaxation based message passing algorithms [5–8] and flow based algorithms [1,9–12]. A recent comprehensive survey about the algorithms for MAP inference can be found in [13].

In message passing based approaches, convergence to the optimal solution is defined only in the limit when the algorithm is allowed to run indefinitely. Even if the algorithm converges, it may not always lead to an integral solution. A popular approach is to first define an LP relaxation of the problem followed by message passing on the relaxed version [5,6,14]. Komodakis and Paragios [15] have proposed a dual decomposition framework which solves the dual relaxation of the general MRF–MAP problem. The original problem is first decomposed into a series of subproblems that are easy to optimize. The solution of the original hard problem is obtained by cleverly combining solutions from the subproblems. Getting the combined solution in a dual decomposition framework is a non-trivial task even when the optimal solutions to subproblems are given.

Flow based algorithms solve a combinatorial optimization problem such as max-flow/min-cut and have been shown to give better performance in practice [16]. Ishikawa [17] has proposed to reduce the higher order potentials into binary ones and combine the reduction with existing flow based algorithms. However, this reduction itself is exponential in the clique size in most cases. The reduction method suggested by Fix et al. [18] has been shown to give better performance theoretically and experimentally. Their technique is to reduce a group of higher order terms at the same time instead of each term individually. A big shortcoming of reduction based approaches is that the reduced pairwise problem is often non-submodular hence hard to solve optimally. This is true even when the original higher order version was submodular. A different approach to reduction is by Kahl and Strandmark [19] using the generalized roof duality. Their method can find a submodular approximation to the original higher order function of clique size at most 4. Their approach produces

* Corresponding author.
E-mail address: dineshk@cse.iitd.ac.in (D. Khandelwal).

http://dx.doi.org/10.1016/j.cviu.2015.10.016
1077-3142/

better solutions in practice but is computationally expensive. Reduction based approaches do not guarantee optimal inference even when the potential function is submodular, for which theoretical algorithms for finding optimal solutions are known [20].

Recently, Arora et al. [1] has proposed a max-flow/min-cut based approach to deal with higher order potentials directly. Their algorithm, called Generic Cuts (GC), can be seen as a combination of the LP relaxation based and the flow based approaches. They define a gadget based flow graph corresponding to Lagrangian dual of LP relaxation, over which running a modified max-flow algorithm results in the optimal solution when the potentials are submodular. Fix et al. [10] have replaced the augmenting path style flow algorithm used in GC with their improved IBFS, showing improvement in running time for clique size of 4. The worst case time complexity of the Generic Cuts is $O(n|C|^2k^3 2^k)$ where n, $|C|$ and k are the number of variables, number of cliques and the size of max-clique, respectively. There are (2^k) labelings on a clique and the higher order potential cost of each labeling contributes a constraint in the dual formulation. So the term $(|C|2^k)$ in time complexity captures the total number of constraints. Though the algorithm has been shown to significantly outperform existing state of the art approaches, it is still exponential in the clique size making it intractable for larger clique sizes.

The hardness in MRF–MAP optimization problem is due to the number of possible labelings on a clique which increase exponentially with clique size. We observe that for many such inference problems in computer vision, many of these possible labelings are forbidden since they attract a high cost in the energy function. In the corresponding optimization problem as defined in Generic Cuts, where the labeling cost maps to the cost of cutting a few edges in a flow graph, the solution obtained depends only on a very small set of constraints defining the cut. To illustrate the point, consider creating a GC flow graph where all but a subset of the constraints are ignored. We call such a problem the *relaxed* problem. The corresponding flowgraph is referred to as the *relaxed* graph. The constraints which are ignored are called the *inactive* constraints and the remaining ones are referred to as the *active* constraints. As we show later in this paper, the flow in the relaxed graph is an upper bound for the flow in the original graph. We also show that a maximum flow in the relaxed graph which does not violate any flow constraint of the original graph is a maximum flow for the original graph as well.

Motivated by the observation mentioned above, we propose *Lazy Generic Cuts* (LGC), which brings in the constraints lazily to the *active* set, and gradually learns the relaxation in which a maximum flow is consistent with the original graph. For finding a maximum flow in the relaxed graph, we use standard GC with the modification to calculate residual capacities using active constraints only. Since the number of active constraints are usually significantly smaller than the total number of constraints in a typical computer vision problem, this allows each iteration to run much faster as well and requiring only a fraction of the memory compared to that of original GC.

We show that the LGC algorithm is guaranteed to terminate in a finite number of steps at the optimum when the clique potentials are submodular. Though there is no optimality guarantee for non-submodular clique potentials, our experiments show that the solutions inferred by our algorithm have good visual quality and their energy value is close to that obtained by GC. Note that the number of active constraints can be significantly less than the total number of constraints. This property is key to scalability of the LGC algorithm to larger clique sizes.

LGC can also be seen in the light of cutting plane inference algorithms proposed earlier for graphical models [21,22]. One of the key characteristics of these algorithms is their ability to deal with large number of constraints by working with a relaxed problem which has only a small subset of constraints (i.e. the active set in our terminology). If at optimality all constraints in the original problem are satisfied, then the current solution is also optimal for the original problem. Otherwise the algorithm refines the active set by including the violated constraints which is then used to cut down the feasible space. In our approach, we use the simple but effective strategy of scanning through all the constraints and including all the violated constraints for the next iteration. After including violated constraints we solve the new relaxed problem. This process is repeated until there are no violated constraints. Exploring more efficient ways for refining the set of active constraints is part of the future work. It should be noted that despite the linear scan to refine the active set, LGC algorithm can still give substantial gains as such scans are only done at the beginning of each iteration. However this allows each flow augmentation of GC to run on a significantly smaller set of active constraints only.

We evaluate our algorithm on binary denoising problem. Our algorithm is able to scale to clique sizes which none of the existing algorithms can handle. Even on problem sizes which earlier techniques could handle, we are significantly more efficient both in terms of running time and memory.

The outline of this paper is as follows. We first present the required background on Generic Cuts [1] in Section 2. We then describe the LGC algorithm in detail in Section 3. We also give a proof of correctness and convergence for the proposed algorithm in this section. We describe our experimental evaluation in Section 4 and conclude the paper with directions for future work in Section 5.

2. Background

The generalized version of MRF–MAP minimization problem with higher order cliques is given as:

$$E(\mathbf{l}_{\mathcal{P}}^*; D, W) = \min_{\mathbf{l}_{\mathcal{P}}} \left[\sum_{p\in\mathcal{P}} D_p(l_p) + \sum_{\mathbf{c}\in\mathcal{C}} W_{\mathbf{c}}(\mathbf{l}_{\mathbf{c}}) \right], \tag{1}$$

where $\mathcal{P}$ denotes the set of pixels/sites and $\mathcal{C}$ denotes the set of cliques. l_p denotes label on a pixel p and $\mathbf{l}_{\mathbf{c}}$ denotes labeling on clique $\mathbf{c}$. $\mathbf{l}_{\mathcal{P}}$ is the labeling configuration on set of pixels $\mathcal{P}$. D_p is generally referred to as the *Data/Singleton* Term, while $W_{\mathbf{c}}$ is referred to as the *Clique Potential*. We follow the notation of Arora et al. [1] and give the LP relaxation of the MRF–MAP minimization problem as follows:

$$\min_{X_p^l, Y_{\mathbf{c}}^{\mathbf{l}_{\mathbf{c}}}} \quad \sum_{p\in\mathcal{P}}\sum_{l\in\mathcal{L}} D_p(l) X_p^l + \sum_{\mathbf{c}\in\mathcal{C}}\sum_{\mathbf{l}_{\mathbf{c}}\in\mathcal{L}^k} W_{\mathbf{c}}(\mathbf{l}_{\mathbf{c}}) Y_{\mathbf{c}}^{\mathbf{l}_{\mathbf{c}}} \tag{2}$$

$$\text{subject to} \quad \sum_{l\in\mathcal{L}} X_p^l = 1, \quad p \in \mathcal{P}, \tag{3}$$

$$\sum_{z\in\{\mathbf{l}_{\mathbf{c}}\}_{p,l}} Y_{\mathbf{c}}^z = X_p^l, \quad p\in\mathcal{P},\ l\in\mathcal{L},\ \mathbf{c}\in\mathcal{C} \tag{4}$$

$$X_p^l \geq 0, \quad Y_{\mathbf{c}}^{\mathbf{l}_{\mathbf{c}}} \geq 0. \tag{5}$$

Here $\{\mathbf{l}_{\mathbf{c}}\}_{p,l}$ denotes the set of all labeling configurations with the label of pixel p as l. We refer to the problem as the primal problem. There are two primal variables introduced in the relaxation. X_p^l has been introduced for each pixel p and for each label x0142; that can be assigned to pixel p. It takes value 1 whenever pixel p is assigned label l and 0 otherwise. Similarly variable $Y_{\mathbf{c}}^{\mathbf{l}_{\mathbf{c}}}$, introduced for each clique $\mathbf{c}$ and each labeling configuration $\mathbf{l}_{\mathbf{c}}$ on clique $\mathbf{c}$, takes value 1 whenever clique $\mathbf{c}$ is assigned labeling configuration $\mathbf{l}_{\mathbf{c}}$. Eq. (3) ensures that each pixel is assigned exactly one label and Eq. (4) enforces consistency between pixel and clique labeling configurations.

The Lagrangian dual of the relaxed LP can be written as:

$$\max_{U} \quad \sum_{p\in\mathcal{P}} U_p \tag{6}$$

$$\text{subject to} \quad U_p \leq h_p^l, \quad p\in\mathcal{P},\ l\in\mathcal{L}, \tag{7}$$

$$\text{where} \quad h_p^l = D_p(l) + \sum_{\mathbf{c}:p\in\mathbf{c}} V_{\mathbf{c},p,l}, \tag{8}$$

$$\sum_{p\in\mathbf{c}} V_{\mathbf{c},p,\mathbf{l}_{\mathbf{c}}^p} \leq W_{\mathbf{c}}(\mathbf{l}_{\mathbf{c}}), \quad \mathbf{c}\in\mathcal{C},\ \mathbf{l}_{\mathbf{c}}\in\mathcal{L}^k, \tag{9}$$

where $\mathbf{l}_{\mathbf{c}}^p$ denote the label of pixel p in labeling $\mathbf{l}_{\mathbf{c}}$.

2.1. Reparameterization

Vectors D, W can be seen as parameters for the primal problem. Similarly vectors U, V together with D, W defines the parameters for the dual problem. Kolmogorov and Rother [23] define *reparameterization* as follows:

Definition 2.1. If two parameter vectors (D, W) and (D', W') define the same energy function, i.e., $E(\mathbf{l}_{\mathcal{P}}; D, W) = E(\mathbf{l}_{\mathcal{P}}; D', W') + \mathcal{F}$ for all configurations $\mathbf{l}_{\mathcal{P}}$ and for a constant $\mathcal{F}$, then (D', W') is called a reparameterization of (D, W).

Intuitively, the above definition says that two parameter vectors correspond to a reparameterization if the respective energy functions differ at most by a constant (positive or negative). Reparameterization is an important concept and has been used extensively in the existing literature [23,24]. Based on the prior work, we explain below some special reparameterizations that will be important for our exposition.

Lemma 2.2 (Primal reparameterization)**.** *The two parameter vectors (D, W) and (D', W') for the primal problem (2) are reparameterizations of each other if for a pixel p, a label l on p, an arbitrary clique $\mathbf{c}$: $p \in \mathbf{c}$, and any δ*

$$D'_p(l) = D_p(l) - \delta,$$
$$\text{and} \quad W'_{\mathbf{c}}(\mathbf{l_c}) = W_{\mathbf{c}}(\mathbf{l_c}) + \delta \quad \forall \mathbf{l_c} : \mathbf{l}^p_{\mathbf{c}} = l \tag{10}$$

It is easy to see that the reparameterization suggested in the lemma simply moves the contribution from the singletons to the clique potential terms for a pixel p which is assigned label l, leaving the sum unchanged. It may be noted that the reparameterization is correct irrespective of the sign of δ.

Lemma 2.3 (Dual reparameterization)**.** *The two parameter vectors (D, W, U, V) and (D', W', U', V') for primal and dual pair (2), (6) are reparameterizations of each other if for a pixel p, a label l on p, an arbitrary clique $\mathbf{c}$: $p \in \mathbf{c}$, and any δ*

$$V'_{\mathbf{c},p,l} = V_{\mathbf{c},p,l} - \delta, \tag{11}$$
$$D'_p(l) = D_p(l) + \delta \tag{12}$$
$$\text{and} \quad W'_{\mathbf{c}}(\mathbf{l_c}) = W_{\mathbf{c}}(\mathbf{l_c}) - \delta \quad \forall \mathbf{l_c} : \mathbf{l}^p_{\mathbf{c}} = l \tag{13}$$

Note that all the dual constraint equations (8) have equal and opposite changes in values of h^l_p and $V_{\mathbf{c},p,l}$. Similarly dual constraints (9) have equal changes on both sides of inequality. This ensures that all the dual constraints remain equivalent. From the primal perspective also, equal and opposite changes have been made in singleton and clique potential terms leaving the sum unchanged (similar to primal reparameterization).

2.2. Generic Cuts

Generic Cuts (GC) proposed by Arora et al. [1] is a flow based algorithm for solving 2-label MRF–MAP problems. For the purpose of the discussion in this paper we will denote the two labels as a and b. GC creates a gadget based flow graph for cliques of arbitrary size. The flow graph contains a gadget corresponding to every clique in the MRF. A gadget corresponding to a clique of size k contains $k + 2$ vertices. A sample flow graph corresponding to a single clique of size 5 is shown in Fig. 1. There are three types of nodes in the flow graph. There is a *pixel* node corresponding to each pixel in the original problem, two *auxiliary* nodes (m and n) for every gadget and two terminal nodes source and sink for the overall graph. The pair of edges from a pixel node, p, to the auxiliary nodes, m and n, is called a *conjugate edge pair* corresponding to pixel p. A pixel node corresponding to pixel p is connected to *terminal* nodes source and sink by edges called *terminal* edges. The capacity of the edge from p to the sink is set as $D_p(a)$ and that of the edge from the source to p is set as $D_p(b)$. Every gadget corresponding to a clique stores a potential table of size 2^k (the number of possible labelings). Finding a maximum flow in the GC flow graph is equivalent to performing the optimal inference for the original problem when the clique potentials are submodular.

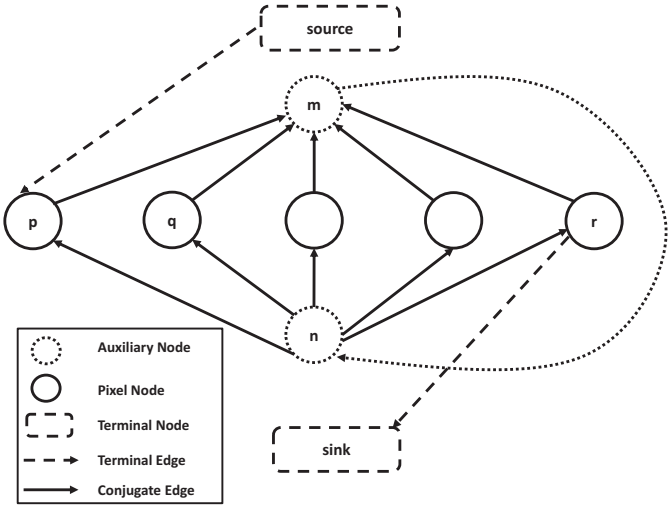


Fig. 1. A gadget for 5-clique.

The gadget in GC models the dual problem as given by Eq. (6) such that there exists a one to one relationship between the dual variables and the flow in the conjugate edge pairs in the gadget as follows:

$$V_{\mathbf{c},p,a} = f^c_{n\to p} - f^c_{p\to m}, \tag{14}$$

where $f^c_{n\to p}$ represents the flow in edge $n \to p$ corresponding to clique $\mathbf{c}$. GC restricts flow to be non-zero in only one edge of a conjugate edge pair and maintains all dual variables of type $V_{\mathbf{c},p,b}$ at zero throughout the execution of the algorithm. Replacing variables of type $V_{\mathbf{c},p,a}$ in Eq. (9) with flow in the conjugate edge pair from Eq. (14), gives a constraint on how much flow can be sent in each edge of the gadget as follows:

$$\sum_{p\in\mathbf{c}:\mathbf{l}^p_{\mathbf{c}}=a} (f^c_{n\to p} - f^c_{p\to m}) \le W_{\mathbf{c}}(\mathbf{l_c}), \quad \mathbf{c} \in \mathcal{C},\ \mathbf{l_c} \in \mathcal{L}^k. \tag{15}$$

Definition 2.4. Using the notation given by Arora et al., each constraint described by Eq. (15) on the flow in the conjugate edge pairs is called a dual feasibility constraint (DFC) corresponding to labeling $\mathbf{l_c}$ on clique $\mathbf{c}$ and is denoted by $DFC(\mathbf{l_c})$.

All the pixels p such that $\mathbf{l}^p_{\mathbf{c}} = a$ are referred to as the pixels participating in $DFC(\mathbf{l_c})$. The corresponding conjugate edges are called the *participating edges*. A DFC limits the sum of the flows in the set of its participating edges. The quantity $W_{\mathbf{c}}(\mathbf{l_c})$ is called the *cost* of the $DFC(\mathbf{l_c})$. There are 2^k DFCs on a clique of size k corresponding to each possible labeling.

Definition 2.5. The quantity

$$S_{\mathbf{c}}(\mathbf{l_c}) = W_{\mathbf{c}}(\mathbf{l_c}) - \sum_{p\in\mathbf{c}:\mathbf{l}^p_{\mathbf{c}}=a} (f^c_{n\to p} - f^c_{p\to m}) \tag{16}$$

is referred to as the slack of the $DFC(\mathbf{l_c})$.

The slack of a DFC is the difference between the sum of flows in the participating edges and the cost of the DFC.

Definition 2.6. A DFC with slack equal to zero is referred to as tight.

Definition 2.7. The residual capacity, $R_{\mathbf{c}}(p)$, of a conjugate edge pair corresponding to a pixel p in clique $\mathbf{c}$ is defined as the minimum slack of all the DFCs in which it participates. For the residual capacity calculation, we exclude the DFCs corresponding to uniform labeling, i.e.,

when either all the pixels have been assigned label a (denoted as $\mathbf{l_c} = \mathbf{a}$) or all the pixels have been assigned label b (denoted as $\mathbf{l_c} = \mathbf{b}$). Formally:

$$R_{\mathbf{c}}(p) = \min_{\substack{\mathbf{l_c}:\mathbf{l}_{\mathbf{c}}^{p}=a,\\ \mathbf{l_c}\neq\mathbf{a},\mathbf{b}}} S_{\mathbf{c}}(\mathbf{l_c}) \tag{17}$$

It is instructive at this point to compare the notion of capacity of an edge in GC with that in a standard max-flow problem. In a standard max-flow problem, the flow in an edge is constrained by a single scalar called capacity of the edge. In contrast, the constraints on the flow in an edge in a GC flow graph are due to multiple DFCs in which that edge participates. Each such DFC limits the sum of flows in the participating edges. The capacity of an edge in GC can be seen as the flow of maximum value that can be sent in the conjugate edge pair without violating any DFC. Any flow augmentation through a gadget increases flow in one pair of conjugate edge, and decreases the same amount in another pair of conjugate edge, so that slacks in DFCs corresponding to uniform labeling never changes and can never be violated. That is why the residual capacity calculation excludes DFCs corresponding to uniform labeling.

It has been known that a MRF–MAP problem with pairwise submodular potentials can be solved by finding a minimum cut in an appropriately constructed graph. It is known that finding maximum flow is the dual problem of minimum cut. Arora et al. showed that with the generalized notion of capacity proposed by them, the dual is essentially finding a maximum flow in the gadget based flow-graph. They showed that almost all the results from the standard max-flow problem such as augmenting path, non-decreasing shortest path length extend to the GC framework as well. This allows them to use any regular max-flow algorithm within the framework for optimizing the dual. Arora et al. give a generalized definition for the cut in the gadget based graph which maps to the primal (integral) problem. They show the optimality of the inference as stated in the below theorem.

Theorem 2.8. *(Arora et al. [1])* In a flow graph when the clique potentials on all the cliques are submodular, the maximum flow value is equal to the cost of a minimum cut and is the optimal inference for the MRF–MAP problem (2).

The output labeling is recovered from a minimum cut by labeling all pixels in the S side of the cut as b and the remaining pixels as a.

A residual graph G' with respect to flow f of a gadget graph G consists of the same vertex set as G and the edge capacities defined as residual capacities in G. The implementation presented in [1] uses an augmenting path algorithm to find a maximum flow. In each iteration of flow augmentation, a shortest path is found between the source and the sink and the flow equal to the minimum capacity of the edge along the path is augmented through the path. After flow augmentation, a residual graph is constructed by updating the slacks of DFCs along the path. The steps are repeated until no more flow can be augmented after which the pixel nodes reachable from sink are given the label a and remaining nodes the label b. Note that flow can be augmented using any heuristic used in traditional max flow algorithms. In practice the algorithm by Boykov and Kolmogorov [9] for augmenting flow works faster compared to other strategies.

3. Lazy Generic Cuts (LGC)

We explain our proposed algorithm in this section. We start with the preliminaries to motivate our approach followed by the details of our algorithm. For the sake of conciseness and readability, the proofs of the lemmas and theorems have been moved to the appendix.

3.1. Preliminaries

Lemma 3.1 relates the residual graph after flow augmentation in the gadget graph and the reparameterization:

Lemma 3.1. *The residual graph created after each flow augmentation in GC corresponds to a reparameterization of the original problem.*

The lemma allows us to visualize the residual graph created after each flow augmentation, in the original graph, as a reparameterization of the original problem. The two problems differ by a constant factor equal to the augmented flow. The GC algorithm can thus be equivalently seen as carrying out a series of reparameterizations. It is interesting to note that the set of reparameterizations carried out by GC keeps all DFCs non-negative at all stages.

Consider a set $\mathcal{W}$, containing all DFCs present in the dual problem (6). Denote an active set of DFCs by $\mathcal{A} \subseteq \mathcal{W}$. The relaxed dual problem is defined by considering only DFCs which are present in the set $\mathcal{A}$:

$$\max_{U} \quad \sum_{p\in\mathcal{P}} U_p \tag{18}$$

$$\text{subject to} \quad U_p \leq h_p^l, \quad p \in \mathcal{P},\ l \in \mathcal{L}, \tag{19}$$

$$\text{where} \quad h_p^l = D_p(l) + \sum_{\mathbf{c}:p\in\mathbf{c}} V_{\mathbf{c},p,l}, \tag{20}$$

$$\sum_{p\in\mathbf{c}} V_{\mathbf{c},p,\mathbf{l}_{\mathbf{c}}^{p}} \leq W_{\mathbf{c}}(\mathbf{l_c}), \quad \mathbf{c} \in \mathcal{C},\ \mathbf{l_c} \in \mathcal{L}^k,\ DFC(\mathbf{l_c}) \in \mathcal{A}. \tag{21}$$

Definition 3.2. A flow graph $G_{\mathcal{A}}$ corresponding to the relaxed dual problem (18), considering DFCs in the set $\mathcal{A}$ only, is called the relaxed graph.

Lemma 3.3. *Let $\mathcal{F}_{\mathcal{A}}$ be a valid flow*[1] in a relaxed graph $G_{\mathcal{A}}$. *The residual graph created after augmenting flow $\mathcal{F}_{\mathcal{A}}$ in G corresponds to a reparameterization of the original problem.*

The proof follows simply from the observation that the number of vertices and the edges are same in G and $G_{\mathcal{A}}$ and therefore the flow in $G_{\mathcal{A}}$ can be mapped to a flow in G as well. The fact that a flow in G corresponds to a reparameterization (Lemma 3.1) is not affected by the choice of ignoring some DFCs while calculating the flow.

We can now relate a maximum flow in $G_{\mathcal{A}}$ and G as follows:

Lemma 3.4. *Consider a relaxed graph $G_{\mathcal{A}}$ and a maximum flow $\mathcal{F}_{\mathcal{A}}$ in it. If $\mathcal{F}_{\mathcal{A}}$ is a valid flow for G, then it is a maximum flow for G also.*

Lemma 3.4 gives us an immediate strategy to find a maximum flow in G. We can search for a relaxed graph $G_{\mathcal{A}}$, such that a maximum flow in $G_{\mathcal{A}}$ is a valid flow in G. Searching for a $G_{\mathcal{A}}$ essentially implies searching for a suitable $\mathcal{A} \subseteq \mathcal{W}$. A simple search strategy could be to start with an arbitrary $\mathcal{A}$ and find a maximum flow $\mathcal{F}_{\mathcal{A}}$ in corresponding $G_{\mathcal{A}}$. If $\mathcal{F}_{\mathcal{A}}$ is not valid for G, then there must exist some DFCs which are violated in G. We can add these DFCs to the set $\mathcal{A}$ and rerun the max-flow algorithm in $G_{\mathcal{A}}$. The process continues until we reach a set $\mathcal{A}$ such that a maximum flow in $G_{\mathcal{A}}$ is a valid flow in G.

A more efficient version of algorithm can reuse the flow computation in the previous iteration. For the case of pairwise problem(clique size 2) Kohli and Torr [24] have suggested to reuse the flow computed in an iteration of alpha expansion [25] as initialization for the next iteration. Since the initialized flow can be invalid for the next iteration, they suggest a reparameterization of the problem such that the flow becomes valid for the reparameterized problem. In our case, the flow in the relaxed graph corresponds to a reparameterization of the original problem (as suggested by Lemma 3.3). A flow $\mathcal{F}_{\mathcal{A}}$ which is not

[1] *A flow $\mathcal{F}$* is called valid if it does not violate any DFC in corresponding dual formulation.

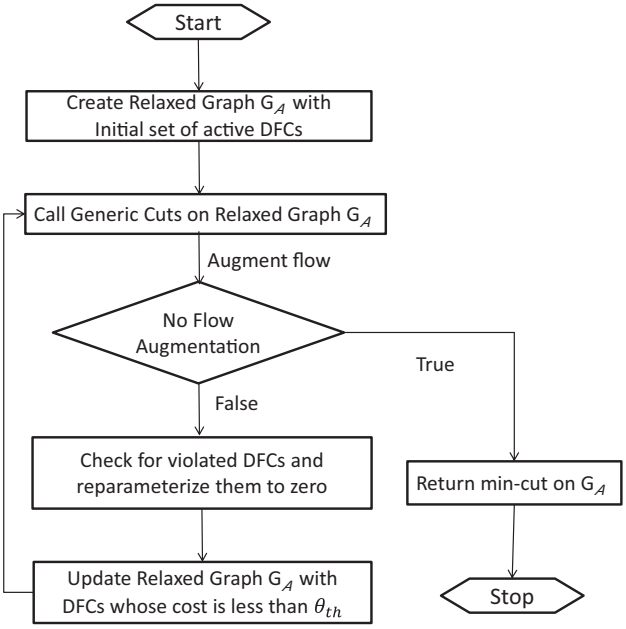


Fig. 2. Flow chart of LGC.

valid for G essentially implies that slacks of some of the DFCs have become negative in the reparameterized problem. This is not an issue since we can always do another reparameterization to make slacks of these DFCs non-negative (we give the details in the next section). The reparameterized problem can now be solved using GC or any other flow algorithm. Note that working with the reparameterized problem allows us to reuse the computation for the DFCs which were not violated in the previous iteration. This forms the broad strategy of our proposed algorithm, *Lazy Generic Cuts (LGC)*, explained in the section below.

3.2. LGC algorithm

We show the broad steps in our algorithm as a flowchart in Fig. 2. Algorithm 1 provides the pseudocode which we describe next. We initialize the algorithm with a subset of DFCs called the active DFCs (Line 2). As explained earlier, this is a subset of DFCs which are expected to become tight during the standard GC flow augmentations. In general, the challenge is how to find a good starting set. In our implementation, we choose the DFCs having the minimum and second minimum cost as our initial active set. We note that more sophisticated strategies could be employed leading to a decrease in the number of iterations and better efficiency of the proposed algorithm. The relaxed graph is then constructed using the initial active set (Line 3).

We then augment flow in $G_{\mathcal{A}}$ using the technique prescribed in GC (Line 6). We have made a modification in GC for our purpose which maintains and computes residual capacity based upon DFCs in $\mathcal{A}$ only. This ability to ignore the inactive constraints during GC flow augmentation is key to time and memory savings obtained by the LGC algorithm.

It may be noted that we do not necessarily find a maximum flow in $G_{\mathcal{A}}$ and instead stop after *subIter* flow augmentations (or when no flow can be augmented, whichever is earlier). This is because the purpose of the iteration is essentially to find a next suitable set of DFCs $\mathcal{A}$ only. Our observation which is inline with other similar works in traditional max-flow [26] is that augmenting the last few flows in the graph takes most of the time. By stopping the flow augmentation earlier we save on compute time without any significant drop in identification of the new active set.

Algorithm 1 Lazy Generic Cuts Algorithm.

Input: G // Input Graph
Input: *subItr* // Number of flow augmentations per iteration
Input: θ_{th} // Threshold to include a DFC in $\mathcal{A}$
1: $\mathcal{W}$ = GetAllDFCs(G);
2: $\mathcal{A}$ = GetInitialActiveDFCs(G); // Include the DFCs with minimum and second minimum costs per clique.
3: $G_{\mathcal{A}}$ = GetRelaxedGraph(G,$\mathcal{A}$);
4: $\mathcal{F} = 0$; // Initialize total flow to zero.
5: **repeat**
6: $\mathcal{F}_{\mathcal{A}}$ = GenericCuts($G_{\mathcal{A}}$, *subItr*);
7: $\mathcal{F} = \mathcal{F} + \mathcal{F}_{\mathcal{A}}$;
8: // Reparameterize G according to the flow augmented in $G_{\mathcal{A}}$
9: G = Reparameterize(G, $\mathcal{F}_{\mathcal{A}}$);
10: (G, Δ) = ReparamViolated(G, $\mathcal{W} \setminus \mathcal{A}$);
11: $\mathcal{F} = \mathcal{F} + \Delta$;
12: // Bring all DFCs with cost less than θ_{th} after reparameterization in $\mathcal{A}$.
13: $\mathcal{A}_{\theta_{\text{th}}}$ = GetLessThanThetaCostDFCs(G, $\mathcal{W} \setminus \mathcal{A}$, θ_{th});
14: $\mathcal{A} = \mathcal{A} \cup \mathcal{A}_{\theta_{\text{th}}}$;
15: $G_{\mathcal{A}}$ = UpdateRelaxedGraph(G,$\mathcal{A}$);
16: **until** ($\mathcal{F}_{\mathcal{A}} == 0$)
17: return (GC-min-cut($G_{\mathcal{A}}$)); // Return a min-cut in $G_{\mathcal{A}}$

Algorithm 2 Reparameterization for violated DFCs.

function ReparamViolated(G, $\mathcal{W} \setminus \mathcal{A}$)
  $\Delta = 0$;
  for all ($DFC(\mathbf{l_c}) \in \mathcal{W} \setminus \mathcal{A}$) **do**
    if ((δ=GetSlack($DFC(\mathbf{l_c})$, G)) < 0) **then**
      // returns a pixel participating in $DFC(\mathbf{l_c})$ such that $l_{\mathbf{c}}^{p} == a$.
      p = GetPixelToReparameterize($DFC(\mathbf{l_c})$,G);
      $D_p(a) = D_p(a) + \delta$;
      for all ($\hat{l}_{\mathbf{c}} \in \mathcal{L}^{|\mathbf{c}|}$) **do**
        if ($\hat{l}_{\mathbf{c}}^{p} == a$) **then**
          $W_{\mathbf{c}}(\mathbf{l_c}) = W_{\mathbf{c}}(\mathbf{l_c}) - \delta$;
        end if
      end for
      $\delta' = D_p(a)$;
      // If unary cost has become negative, make it zero as well
      if ($\delta' < 0$) **then**
        for all ($l \in L$) **do**
          $D_p(l) = D_p(l) - \delta'$;
        end for
      end if
      $\Delta = \Delta - \delta'$;
    end if
  end for
  return (G, Δ)
end function

Algorithm 3 Get the slack of $DFC(\mathbf{l_c})$.

function GetSlack($DFC(\mathbf{l_c})$, G)
  $slack = W_{\mathbf{c}}(\mathbf{l_c})$;
  for all ($p \in \mathbf{c} : \mathbf{l}_{\mathbf{c}}^{p} == a$) **do**
    $slack = slack - (f_{n\to p}^{c} - f_{p\to m}^{c})$;
  end for
  return($slack$);
end function

Since the numbers of vertices and edges in G and $G_{\mathcal{A}}$ are same we can map the flow in $G_{\mathcal{A}}$ to G. As suggested in Lemma 3.3, the flow augmentation in the graph G can be seen as a reparameterization of the original problem (Line 9). The reparameterization maintains all active DFCs to be non-negative but other (inactive) DFCs may become negative after reparameterization. We refer to such negative cost DFCs as violated constraints. Once we have identified the violated constraints, we reparameterize the problem to bring the cost of such constraints to zero (Line 10).

Our strategy of maintaining an active set is on the basis of the costs of the DFCs. Since the flow augmentation and reparameterization may change these costs, we update the active set at the end of each iteration and compute the new relaxed graph based on the modified active set (Line 15). We fix a threshold θ_{th} and bring all the DFCs, whose cost is less than or equal to this threshold, in the active set (Line 13). GC is now run on the new active set of constraints. The algorithm terminates when the flow augmentation returned by the GC is equal to zero. The min-cut (or max-flow) in the relaxed graph at the termination is output as a solution to the original problem. We give the proof of correctness and convergence of the algorithm in the next section.

3.3. Convergence and correctness

In every iteration of LGC we add at least one constraint to the active set. Since there are finitely many constraints in the problem, the LGC must converge in a finite number of iterations.

Intuitively, the algorithmic steps in LGC can be seen as a specialization of the cutting plane strategy [27]. Initially, we can think of our optimization problem to be the one with the original dual objective function but with only a small subset of the dual feasibility constraints (DFCs). At each iteration of LGC, we check if the solution to the current problem satisfies the required constraints (i.e. satisfies all the original DFCs). If it does, we terminate. If it does not, we impose additional cutting constraints by adding new DFCs to the current problem. Addition of the new constraints does not exclude any feasible solutions of the original problem but reduces the feasible solution region with respect to the current problem. This similarity between LGC and the cutting plane strategy can be used to visualize many of the theoretical guarantees, including convergence and correctness, for the LGC as described below.

We now state the formal results for the convergence and correctness of LGC. The proofs of these claims have been given in the appendix.

Theorem 3.5. *Let G be a flow graph. Lazy Generic Cuts algorithm, as described in Algorithm 1, terminates in a finite number of iterations.*

Theorem 3.6. *Consider a problem containing submodular clique potentials and let $\mathcal{A}$ be the set of active constraints. Let $G_{\mathcal{A}}$ be the relaxed graph at the termination of the LGC algorithm (Algorithm 1). Let $\mathcal{F}$ be the accumulated flow and $\mathcal{E}_{\mathcal{A}}$ be a minimum cut in $G_{\mathcal{A}}$. Then, $\mathcal{F}$ is a maximum flow in G. Further, the set $\mathcal{E}_{\mathcal{A}}$ is a minimum cut in G and has the value $\mathcal{F}$.*

4. Experiments and results

We have extensively evaluated the performance of our proposed approach Lazy Generic Cuts (LGC) on the problem of binary image denoising. We have compared LGC with the current state of the art methods using a variety of clique potentials. Next, we present the details of the methods compared, our dataset, clique potentials and the setup used in our experiments. This is followed by the presentation of our experimental results.

4.1. Experiment setup

All the experiments have been conducted on a computer with 3.1 GHz Core i7 processor with 16 GB of RAM, running the Ubuntu 15.04 operating system. We compare the following four algorithms in our experiments.

- **LGC:** Lazy Generic Cuts (LGC) is the algorithm proposed in this paper. It is built upon publicly available implementation of GC [1] (details below).
- **GC**: Generic Cuts (GC) is the flow based algorithm proposed by Arora et al. [1] for inference with higher order clique potentials. We used the publicly available code[2] for GC.
- **SoS-IBFS**: This is the algorithm proposed by Fix et al. [10] where they have modified the incremental breadth first search (IBFS) algorithm for minimizing submodular functions to work with the sum of submodular (SoS) case. We used the publicly available code for SoS-IBFS.[3] We refer to the algorithm as IBFS in our discussion.
- **ELC-Approx**: This is the reduction based method proposed by Ishikawa [28]. Their method tries to reduce higher order terms into pairwise without introducing any auxiliary variables. We used the publicly available code for ELC-Approx.[4] We refer the work as ELC in this section.

4.1.1. LGC settings

We work with a modified implementation of GC which takes as input the current set of active DFCs and the stopping criteria i.e. number of augmenting flow iterations after which algorithm should be stopped. This is the *subItr* parameter as described in Section 3. We use a value of 30, 000 for the *subItr* parameter in all our experiments. We initialize the active DFC set by choosing the minimum and second minimum cost DFCs, in each clique. We have experimented with two different values of the θ_{th} parameter i.e. $\theta_{\text{th}} = 20$, and $\theta_{\text{th}} = 40$. Recall that θ_{th} controls which DFCs should be added to the active set after the current reparameterization. All the DFCs whose reparameterized cost becomes less than or equal to θ_{th} are made active. We also analyze sensitivity of LGC with varying values of *subItr* and θ_{th}.

All the above mentioned codes are available in C++. We plan to release our code also in C++ under open source license.

4.1.2. Dataset and clique potentials

All our experiments have been performed on the two label image denoising task with higher order cliques. The images used in the experiments have been taken from [29]. In the description below, when we say we use a clique size of $k \times k$, we include potentials over all the overlapping windows of size $k \times k$ with a stride of 1 in the image. We have used different clique potential types in order to show the resilience of our approach with respect to the underlying potential. Specifically, the following clique potentials have been used:

- **Edge based potentials:** The cost of a clique configuration is given by $\sqrt{E}$. Here, E denotes the set edges produced by a labeling configuration. An edge is defined over a pair of neighboring pixels (we consider 4-neighborhood, i.e., up, down, left and right) with opposite labels (a/b) assigned to them. Fig. 3 shows an example. The total number of edges in the figure is 9 and hence, the cost of the configuration is given by $\sqrt{9}$. Edge based potentials have been used previously in the literature [1] and have been shown to perform particularly well for the binary image denoising task. The potential is submodular for the cliques of sizes less than equal to 4 but not in general.

[2] http://www.iiitd.edu.in/~chetan/abstracts/gc.html.
[3] http://www.cs.cornell.edu/~afix/Software/sum-of-submodular.tar.gz.
[4] http://www.f.waseda.jp/hfs/ELC1.04.zip.

Table 1
Comparison of time and memory of different inference algorithms on count based potential with increasing clique size. The potential is submodular for all clique sizes. Image size is 80 × 80. For LGC $\theta_{th} = 40$.

Clique size	Time (s)				Memory (MB)			
	GC	LGC	IBFS	ELC	GC	LGC	IBFS	ELC
2 × 2	0.01	0.06	0.12	0.03	25	37	30	43
3 × 2	0.03	0.21	0.34	0.06	28	64	54	44
3 × 3	0.82	2.53	3.29	2.10	40	233	106	44
4 × 3	92.08	20.9	131.00	80.82	111	344	319	45
4 × 4	4489.85	547.90	701.67	–	1750	580	3104	–

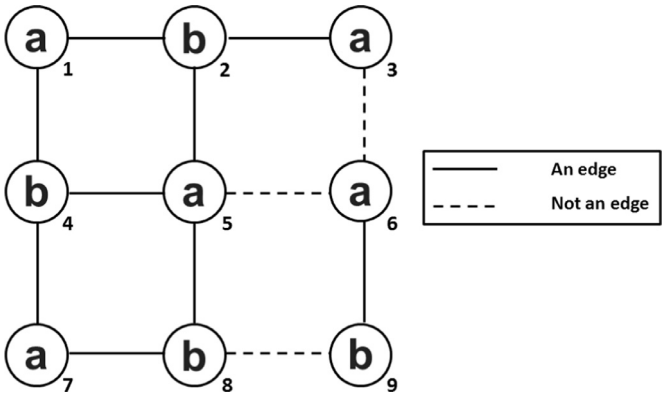


Fig. 3. The figure displays an image patch corresponding to a 3 × 3 sized clique. The nodes are numbered from 1 to 9. Each node in the figure has the label a or b denoting a specific labeling configuration. We consider the 4-neighborhood i.e. up, down, left and right. For example, the neigbhors of node 5 are given as 2, 4, 6 and 8. A pair of neigboring pixels defines an edge if one of them has the label a and the other has label b. The edges for the above figure are shown using solid lines. The total number of edges is $|E| = 9$. Using the edge based potential, the cost of labeling is given by $\sqrt{|E|} = \sqrt{9} = 3$. If $|V_a|$ and $V_b|$ denote the number of pixels labeled a and b, respectively, the cost of count based potential for the above example is given by $|V_a| * |V_b|$ i.e. $5 * 4 = 20$.

- **Count based potentials:** The cost of a clique configuration is given as $|V_a| * |V_b|$ where V_a and V_b denote the sets of pixels labeled a and b, respectively, in the given configuration. For the example shown in Fig. 3, the values of $|V_a|$ and $|V_b|$ are 5 and 4, respectively. The cost of the clique configuration is given by 20. The potential is submodular for all clique sizes and has been used previously in the literature for tasks such as binary object segmentation [30].
- **Learned potentials:** We have also experimented with submodular potentials learnt using the technique proposed by Fix et al. [10]. We have used the publicly available code [31] (with minor modifications to make it work on our system) to generate these potentials. The method uses a set of noisy images and their corresponding ground truth for the learning. Since the original dataset used in [10] is not available, we have generated our own dataset using the method described in [10]. Specifically, we selected 10 butterfly images from the binary image dataset [29] each of size 120 × 120 for the learning. To get noisy images we have added independent Gaussian noise at each pixel and used Hamming distance as a loss function between the ground truth and the predicted image.

In all of our experiments, the unary potentials are chosen as the difference of the pixel intensity value from the respective ideal value (i.e. 0 for label a and 255 for label b) for the given configuration.

4.2. Scalability with clique size

In this section, we present the comparison of LGC with GC, IBFS and ELC using various clique sizes and a fixed image size. Table 1 compares the time and memory required by the various algorithms when the image size is fixed at 80 × 80. The clique potential used is count based. For clique size up to 3 × 3, GC performs best both in terms of time and memory. But its performance degrades significantly as clique size increases. ELC performs reasonably well till clique size of 4 × 3 but is unable to run for larger clique sizes. A '–' in the table means that the algorithm ran out of memory.

At smaller clique sizes LGC is comparable to other algorithms in time and slightly worse in its memory performance. This is because of the extra overhead required to maintain the active set. At larger clique sizes, it takes over other algorithms both in terms of time and memory. At clique size 4 × 4, LGC is the fastest followed by IBFS which is about 1.3 times slower. GC is 8 times slower than LGC. At clique size of 4 × 4, LGC has a third of the memory required by GC and a fifth of the memory required by IBFS. All the algorithms other than LGC failed to scale beyond clique size 4 × 4. We present the scaling behavior of LGC at larger clique sizes later in Section 4.4.

Table 2 (left half) compares the time performance of LGC with the other algorithms using the non-submodular edge based potentials (Section 4.1.2). We observe similar behavior for LGC and GC as observed using count based potentials. ELC's performance degrades significantly with increasing clique size. Interestingly, IBFS has a somewhat better timing behavior compared to LGC (as well as other algorithms). A careful analysis reveals this is because IBFS uses a quick but crude submodular approximation to the original function. The faster inference in IBFS comes at the cost of significantly worse energy values compared to both GC and LGC. In contrast, despite the potential being non-submodular, both GC and LGC are able to obtain a reasonable solution. Fig. 4 compares output of the three algorithms on an image of size 80 × 80 using a 4 × 4 potential validating our thesis above. We give detailed energy comparison for such potentials in the appendix.

Table 2 (right half) compares the memory requirements of the four algorithms for edge based potentials. For lower clique sizes, LGC is slightly worse than GC for the reasons explained earlier. For clique size of 4 × 4, LGC has the best memory performance followed by GC and IBFS, respectively. ELC performs the worst. LGC requires about half the memory required by GC and about a third required by IBFS. There is an interesting observation to make. For GC and IBFS the memory requirements are almost identical to the case of count based potentials for the same clique size. On the other hand, LGC's requirement varies based on the kind of potential chosen (compare results in Tables 1 and 2). This is because of the fact that LGC's memory needs depend on the actual constraints brought in memory unlike GC and IBFS which consume a fixed memory for a given image/clique size combination. Further exploring the connection between LGC's memory requirements and the specific kind of potential used is a direction for future work.

In order to examine the generalizability of LGC's performance, we have also experimented with learned potentials using the approach of Fix et al. [10] (see Section 4.1.2). Because of the limitation of the learning algorithm, we were able to learn the potentials only up to size of 3 × 3. The relative performance of various algorithms is sim-

Table 2
Comparison of time and memory for different inference algorithms on edge based potential with increasing clique size. It may be noted that the potential is submodular for clique size 2×2 only. Image size is 80×80. For LGC $\theta_{\text{th}} = 40$.

Clique size	Time (s)				Memory (MB)			
	GC	LGC	IBFS	ELC	GC	LGC	IBFS	ELC
2×2	0.01	0.08	0.14	0.06	26	37	31	46
3×2	0.06	0.74	0.36	1.45	28	47	60	58
3×3	4.04	11.23	2.96	195	40	93	107	198
4×3	77.73	76.86	29.38	18,533.43	134	238	320	2128
4×4	4082.36	1347.42	700.50	–	1786	1049	3110	–

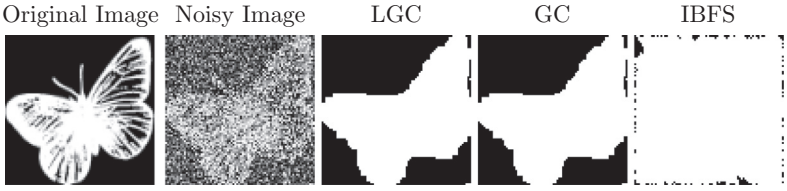
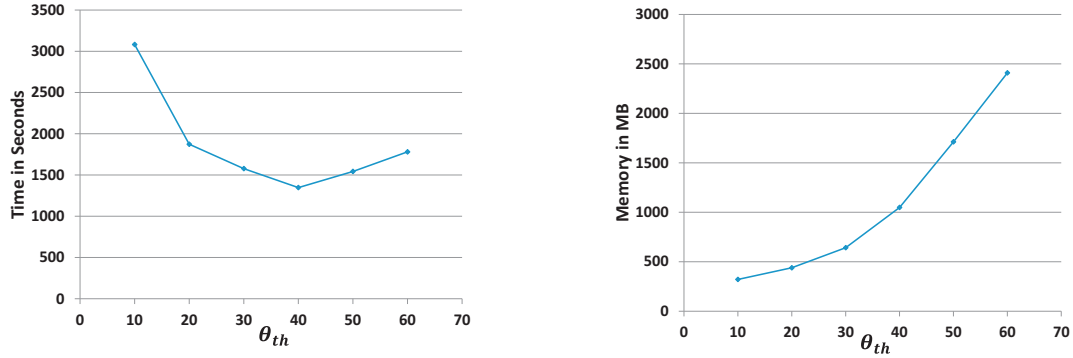


Fig. 4. Comparison of visual quality using nonsubmodular clique potentials for denoising problem. Image size is 80×80 and clique size is 4×4.



(a) Time vs θth (b) Memory vs θ_{th}

Fig. 5. Time and memory variation for LGC with θ_{th} for image of size 80×80 and clique size 4×4.

ilar to the case of submodular count based potentials. We give the detailed comparison in the appendix.

4.3. Scaling with image size

Next we analyze LGC's performance on varying image sizes for a fixed clique size of 4×4. Since we could not learn the potentials beyond size 3×3, we used only count based and edge based potentials for these experiments. ELC fails to scale beyond clique size of 4×3 and is not included in these experiments. The experiments in this section have been performed using count based potentials.

Table 3 presents the time as well as memory required by GC, IBFS and LGC as we vary the image size. LGC has the best running time performance followed by IBFS. GC is significantly slower than both the algorithms. LGC is up to 1.5 times faster than IBFS and close to an order of magnitude faster than GC at all image sizes. In terms of memory, LGC is the best performer with its memory requirement being about a third of GC and about a sixth of IBFS. IBFS is the worst performer in terms of memory.

For non-submodular problems, the behavior is similar to the one observed while analyzing clique size scalability. IBFS runs faster but at the cost of significantly degraded image output quality. We give the detailed analysis in the appendix.

Table 3
Comparison of time and memory of different inference algorithms on count based potential with increasing image size. Clique size is 4×4. For LGC $\theta_{\text{th}} = 40$.

Image size	Time (s)			Memory (MB)		
	GC	LGC	IBFS	GC	LGC	IBFS
40×40	2298.02	92.93	163.52	494	170	730
60×60	3923.84	271.26	390.30	1059	371	1700
80×80	4489.85	547.90	701.67	1750	580	3104
100×100	8297.57	901.32	1395.00	2727	810	4914
120×120	12,961.81	1505.05	1705.10	3802	1178	7018

4.4. Effect of algorithm parameters

Having established the superior performance of LGC over existing state of the art algorithms, we next examine additional properties of LGC such as its scaling behavior beyond clique size of 4×4 where all other algorithms fail to run, the relative amount of time taken by initialization, reparameterization and actual flow computation and the total number of DFCs actually activated by LGC for a given problem. We are also interested in analyzing the effect of the threshold parameter (θ_{th}) and the number of flow augmentations (*subIter*) parameter on time, memory and the energy values obtained (for non-submodular potentials). We analyze these aspects in this

Table 4
Inference time in seconds and memory in MB with increasing clique size. Image size is 40 × 40. $\theta_{th} = 20$ for LGC.

Clique size	LGC	
	Time (s)	Memory (MB)
3 × 3	1.41	35
3 × 4	15.16	50
4 × 4	238.63	104
4 × 5	5013.35	272
5 × 5	246,484.23	1402

Table 5
Inference time required by different components of LGC algorithm on 4 × 4 clique problem with $\theta_{th} = 40$.

Image size	Time (s)			
	Initialization	Flow	Reparameterization	Total time
20 × 20	0.05	19.84	27.07	46.96
40 × 40	0.25	91.42	113.19	204.86
60 × 60	0.59	256.51	326.32	583.42
80 × 80	1.08	609.32	737.02	1347.42
100 × 100	1.72	896.87	1610.13	2510.72
120 × 120	2.52	1564.35	2897.55	4464.42

section. All the experiments in this section have been performed using edge based potentials.

Scaling beyond 4 × 4: Table 4 shows the performance of LGC as we vary the clique size for a fixed image size of 40 × 40. LGC can easily scale to clique sizes of 5 × 5 which is an important milestone for computer vision applications given that the earlier best reported in the literature is clique sizes of 4 × 4.[5] The memory required for clique size 5 × 5 is close to 1.5 GB which is easily accessible on a standard laptop. Time required does shoot up at this clique size and optimizing this further is a part of the future work.

Runtime profiling: We would like to understand where LGC spends most of its time. The key steps in the algorithm are: initialization, reparameterization and the flow computation. Recall that LGC needs to initialize the graph once in the beginning and then the reparameterization and flow computation need to be performed at every LGC iteration (see Algorithm 1). Table 5 compares the total time spent by LGC during the initialization, reparameterization and flow computation steps for varying image sizes and clique size fixed to 4 × 4. Total time taken by LGC is also shown for reference. Initialization cost is negligible in all cases. Interestingly, LGC spends a significant amount of time (more than 50%) during the reparameterization step. Coming up with better strategies to reduce the time spent in reparameterization is a direction for future work.

θ_{th}, no. of iterations and active DFCs: Table 6 shows the number of iterations taken by LGC to converge and also the percentage of the DFCs which have become active at the time when the algorithm converges. We show these numbers for two different values of the θ_{th} parameter: $\theta_{th} = 20$ and $\theta_{th} = 40$. At a smaller value of θ_{th} (20 vs 40), LGC is more conservative in making DFCs active. This results in a larger number of iterations required for the algorithm to converge (41 vs 24) before all the required DFCs are actually made active. At the same time, this results in lesser percentage of DFCs being made active (0.92 vs 2.50). The behavior is similar to typical precision vs recall characteristics of a prediction technique.

[5] There are techniques such as the one proposed by Kohli et al. [32] which scale to larger clique sizes but they use only a very small subset of potential values during inference.

Table 6
Number of LGC iterations and fraction of DFCs in active set at convergence. Clique size is 4 × 4.

Image size	# of GC iterations		% of DFCs active	
	($\theta_{th} = 20$)	($\theta_{th} = 40$)	($\theta_{th} = 20$)	($\theta_{th} = 40$)
20 × 20	13	8	0.98	2.90
40 × 40	12	10	0.86	2.60
60 × 60	17	11	0.92	2.50
80 × 80	23	13	1.00	2.70
100 × 100	31	19	0.91	2.70
120 × 120	41	24	0.92	2.50

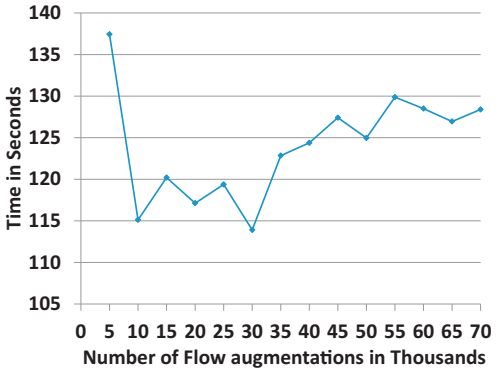


Fig. 6. Time taken by LGC for various values of flow augmentations allowed in each LGC iteration. Image size is 60 × 60, clique size is 4 × 3 and $\theta_{th} = 20$.

We have performed a detailed analysis of the performance of LGC as we vary the threshold parameter in a larger range for a given image and clique size. As the value of θ_{th} increases, the chance of a DFC being included in the active set increases. Some of these DFCs belong to the minimum cut we are trying to find and the chances of such DFCs being included in the active DFCs set quickly increase with increase in θ_{th}. This reduces the number of LGC iterations and therefore improves the time efficiency. Increasing the threshold to a very high value is not useful since this essentially turns LGC to GC, having only one iteration but spending wasteful time in finding residual capacity with constraints which will not be included in minimum cut. We, therefore, expect to see a sweet spot for the threshold giving us maximum benefit in efficiency. Fig. 5(a) confirms the same. On the other hand, the memory requirement is expected to monotonically increase with increasing θ_{th}. Fig. 5(b) confirms the expected behavior.

Effect of parameter subIter: Fig. 6 shows the variations of time required for LGC as we change the number of flow augmentations in each iteration of LGC. The image size for the experiment is 60 × 60 and clique size is 4 × 3 with $\theta_{th} = 20$. The experiment spells the reason for the choice of $subIter = 30,000$ in all our experiments.

4.5. Visual quality

The purpose of the experiments here is to validate the algorithmic improvement suggested in the paper. So far in the paper we have focused on the computational efficiency of our algorithm and the energy of the inferred solution. In this section we show the improvement in visual quality with increase in the clique size. Fig. 7 shows the visual results obtained for denoising different images of size 80 × 80 as the clique size is increased from 2 × 2 to 4 × 4.

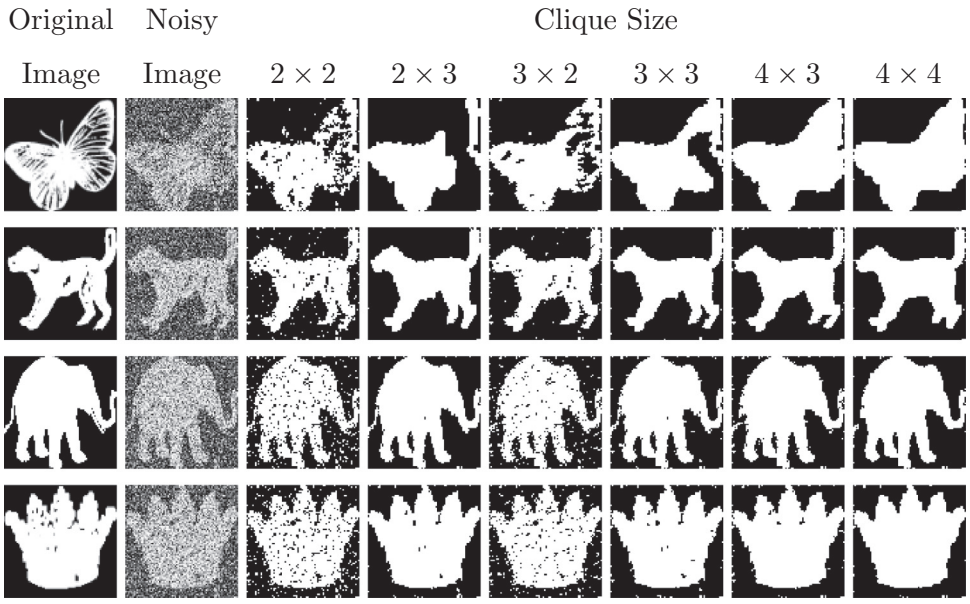


Fig. 7. Improvement in visual quality for denoising problem using cliques of different sizes. Image size is 80 × 80.

5. Conclusion and future work

In this paper, we have proposed a lazy version of the state of the art algorithm, Generic Cuts (GC), to solve the MRF–MAP problem. Our algorithm, called the Lazy Generic Cuts (LGC), exploits the fact that for many vision problems there is only a small number of constraints which are tight in the final solution and therefore, most of the constraints do not matter for deciding the min-cut/max-flow. We express the flow based moves in GC as a set of reparameterizations. Our algorithm works by running GC iteratively using the current set of active constraints. This set is gradually increased at every iteration by including those constraints which become tight (or near tight) based on the last GC run. LGC is guaranteed to give the same results as GC when the potentials are submodular. Our experiments clearly demonstrate that LGC can significantly outperform GC as well as other state of the art algorithms both in terms of time and memory on binary denoising problem. Further, LGC can scale to problems with clique size 5×5 which none of the existing algorithms are able to.

One of the directions for future work includes working in a memory bound scenario where we are given an upper bound on the available amount of memory. Can we then selectively decide which constraints to include in active set and which ones to take out such that we are always within the memory bound (note that the current version of LGC never removes a constraint from the active set)? What kind of convergence guarantees can be given in such a scenario? Another direction for further work includes establishing a deeper connection with cutting plane style algorithms. Most of these algorithms do not need to look at the entire constraint set to find a constraint which is violated by the current solution. Can we do something similar enabling us not to scan the entire set to check for violated constraints? It is an interesting problem to pursue in future work. Other directions of research include solving other classes of problems with techniques similar to our, e.g. multi-label problems [4,11]. It would be interesting to see if we can use LGC for learning the higher-order potential function inside structured learning approach of Fix et al. instead of their incremental breadth first search (IBFS) algorithm.

Appendix A. Proof of lemmas

A.1. Proof of Lemma 3.1

First consider a simple case of flow of type $source \rightarrow p \rightarrow sink$. The residual graph after flow of this type does not affect any DFC. However, the residual capacity of the terminal edges is decreased by δ. The maximum flow value in original and residual graphs differs by δ. Now consider the reparameterization where both the data costs $D_p(a)$ and $D_p(b)$ have been reduced. The flow graph for the reparameterized problem is identical to the residual graph. Similarly, the reparameterized and the original problems differ by the value δ for all labeling configurations.

Now consider flow through paths of length 1. We use the definition of path length in terms of number of path fragments, as defined by Arora et al. [1], where a path fragment is a portion of augmenting path containing a pair of pixel nodes and the auxiliary nodes from a clique/gadget containing the pair. Consider a path of type $source \rightarrow p \rightarrow q \rightarrow sink$. Any flow augmentation of δ through such a path decreases the residual capacities of terminal edges and changes slack of DFCs in the following way. For node q, the slacks of all DFCs, in which the edge corresponding to q participates, decrease by δ. For node p, the slacks of all DFCs, in which the edge corresponding to p participates, increase by δ.[6] The reparameterized problem can be understood from Lemma 11 where two reparameterizations have been done, one with respect to p of $-\delta$ and another with respect to q of $+\delta$.

The paths of length more than 1 can be assumed to be of type $source \rightarrow p \rightarrow q \rightarrow r \rightarrow sink$, containing one or more nodes of type q. The reparameterized graph to residual graph equivalence can be understood by splitting the path into path fragments and establishing the equivalence for each path fragment. In this case the equivalence can be established for path fragments $source \rightarrow p \rightarrow q$ and $q \rightarrow r \rightarrow sink$ using the similar arguments as for paths of length 1.

Therefore residual graph after any flow augmentation in GC can be seen as a flow graph corresponding to the reparameterized version of the original problem where augmented flow can be seen as the difference in cost for a labeling between the original and reparameterized problems.

[6] As a corollary, for the DFCs which contain edges corresponding to both p and q, the slack does not change. However, this has no consequence on our current discussion.

Note that the argument presented above does not depend upon the value of the flow or if the flow violates any DFC. The same argument can therefore be used to prove the reparameterization view for the relaxed graph as well, when the flow is computed ignoring (and possibly violating) some of the DFCs (Lemma 3.3).

A.2. Proof of Theorem 3.5

Consider an LGC iteration where the set of active constraints is given by $\mathcal{A}$. Let the corresponding relaxed graph be denoted by $G_{\mathcal{A}}$. Let $\mathcal{F}$ be the flow when GC is called with $\mathcal{A}$ as the active set of constraints. The LGC algorithm (Algorithm 1) terminates if the value $\mathcal{F}_{\mathcal{A}}$ of the flow is equal to zero. We first consider the simpler case, when $sublter = \infty$ and for each iteration we can augment as much flow as possible. This will mean that $\mathcal{F}$ is a max-flow for the relaxed graph. It suffices to prove that for every LGC iteration, we add at least one constraint to the active set. This is because the active set is bounded above by the total number of constraints and proving above will mean that the active set becomes the entire set of constraints in a finite number of iterations. Running GC with this set will give a max-flow in the original graph (since all the constraints are active). Hence, the net flow in the residual graph will be zero in the next iteration and the algorithm will terminate.

Let $\mathcal{W}_{\mathcal{V}} \subseteq \mathcal{W} \setminus \mathcal{A}$ be the subset of inactive constraints which are violated by the flow $\mathcal{F}$ obtained in $G_{\mathcal{A}}$. Clearly, $\mathcal{W}_{\mathcal{V}}$ is non-empty, since otherwise LGC will terminate in the next iteration (the flow in the residual graph from the last step would be zero). At the end of each iteration, we reparameterize to an equivalent problem where all such violated DFCs are reparameterized to zero. Hence, in the next step when all the constraints whose costs are less than equal to θ_{th} ($\theta_{\text{th}} \geq 0$) are made active (Line 10, Algorithm 1), we can safely assume that at least one constraint is added to the active set. Hence, the active set increases monotonically at every iteration.

The above arguments suffice to prove the convergence when we augment a maximum flow in the relaxed graph. In practice we allow only a certain number of flow augmentations in each iteration. However, such a change does not effect the convergence guarantees. In any flow graph including gadget based flow graph, the number of flow augmentations is bounded by n^2 (n is the number of nodes/pixels). With the bound on number of flow augmentations, in each iteration of LGC either the set of active constraints becomes bigger or non-zero number of flow augmentations have happened. Since both the quantities are finite, the total number of LGC iterations is also finite.

By the arguments presented above, LGC always terminates in finite iterations.

A.3. Proof of Lemma 3.4

The data terms $D_p(l)$ are identical in $G_{\mathcal{A}}$ and G and the DFC costs $W_{\mathbf{c}}(\mathbf{l_c})$ in $G_{\mathcal{A}}$ are greater than or equal to those in G (by construction). Hence, any flow $\mathcal{F}$ in G, which does not violate any DFCs in G, also does not violate any DFCs in $G_{\mathcal{A}}$. Hence, $\mathcal{F}$ is a valid flow for $G_{\mathcal{A}}$ also. Since, any flow F in G is also a valid flow in $G_{\mathcal{A}}$, the value of maximum flow in G must be less than or equal to the maximum flow value in $G_{\mathcal{A}}$.

A.4. Proof of Theorem 3.6

Lemma 3.3 shows that the residual graph obtained after flow augmentation in a relaxed graph corresponds to a reparameterization of the original problem. At the end of each iteration we perform a reparameterization to bring the cost of the violated constraints to zero. Therefore, the graph obtained at the end of each iteration as well as at the termination of LGC corresponds to a reparameterization of the original problem. Such a reparameterized problem differs from the original problem by the constant $\mathcal{F}$. By construction, there is no flow augmentation possible in the reparameterized problem at the termination. Therefore, the value of maximum flow is equal to the cost of a minimum cut, which is equal to zero. Note that a reparameterization of the problem preserves the set of edges on a minimum cut in the corresponding graph. Therefore, the minimum cut in the graph at the termination of LGC is the same as the minimum cut in the original graph. Furthermore, such a cut will take the value $\mathcal{F}$ in the original graph. The theorem states the same.

Appendix B. Detailed experimental results

B.1. Comparison on non-submodular clique potential

We have compared LGC for non-submodular problem using edge based potentials as described before. We observe similar behavior for LGC and GC as in count based potentials. Interestingly, IBFS seems to use quick but crude submodular approximation resulting in faster inference compared to other algorithms but also substantially worse energy values compared to GC and LGC. On the other hand, despite the potential being non-submodular, both GC and LGC are able to obtain a reasonable solution. Table B.7 compares the energy values obtained by all the four algorithms for varying clique sizes.

Table B.8 presents the results as we vary the image size (for a fixed clique size). Table B.9 presents the energy values obtained by the three algorithms validating our claim about poor quality of results obtained by IBFS. Fig. 4 already compared the image quality obtained by the three algorithms on an image of size 80×80 using clique size 4×4. Table B.8 also depicts the memory requirements of the three algorithms with varying image size. LGC is the best performing algo-

Table B.7
Comparison of minimum energy value reached by different inference algorithms on edge based potential with increasing clique size. Image size is 80×80 and $\theta_{\text{th}} = 40$ for LGC.

Clique size	Energy value			
	GC	LGC	IBFS	ELC
2×2	584,511	584,511	584,511	636,423
3×2	610,638	610,638	610,641	754,210
3×3	631,995	631,995	632,053	755,161
4×3	643,248	643,248	642,985	755,412
4×4	659,652	659,652	871,406	–

Table B.8
Comparison of time and memory for different inference algorithms on edge based potential with increasing image size. Clique size is 4×4. For LGC $\theta_{\text{th}} = 40$.

Image size	Time (s)			Memory (MB)		
	GC	LGC	IBFS	GC	LGC	IBFS
40×40	2169.13	204.86	161.40	512	240	1428
60×60	3684.62	583.42	379.43	1035	513	3371
80×80	4082.36	1347.42	700.50	1786	1049	3110
100×100	6049.85	2510.72	1150.22	2693	1553	4916
120×120	11,301.17	4464.42	1604.04	3849	2213	7158

Table B.9
Comparison of minimum energy value reached by different inference algorithms on edge based potential with increasing image size. Clique size is 4×4. For LGC $\theta_{\text{th}} = 40$.

Image size	Energy value		
	GC	LGC	IBFS
40×40	173,530	173,530	213,649
60×60	380,373	380,372	491,084
80×80	659,652	659,652	871,406
100×100	1,009,635	1,009,627	1,320,299
120×120	1,431,658	1,431,663	1,877,654

Table B.10
Comparison of inference time and memory for different inference algorithms on learned potential in seconds with increasing clique size. Image size is 80 × 80 and $\theta_{th} = 40$ for LGC.

Clique size	Time (s)				Memory (MB)			
	GC	LGC	IBFS	ELC	GC	LGC	IBFS	ELC
2 × 2	0.01	0.09	0.13	0.06	26	38	28	43
3 × 2	0.04	0.19	0.29	0.13	27	58	40	44
2 × 3	0.13	0.25	0.27	0.13	27	57	40	44
3 × 3	0.96	4.28	4.10	217.67	40	279	103	200

rithm with its memory requirement being up to half that of GC and between a third and a fifth of IBFS (for different image sizes). IBFS performs the worst of the three algorithms in terms of memory requirements.

B.2. Comparison using learnt potentials

Table B.10 compares the time and memory required by all the four approaches using learned potentials on an image size of 80 × 80 with varying clique sizes. We were unable to learn the potentials of size greater than 3 × 3.[7] The relative performance of various algorithms are similar to those obtained for submodular count based potentials. The real advantage of LGC comes at higher order potentials which we are not able to demonstrate here due to the limitation of the learning algorithm. Memory required by all the four algorithms with varying clique sizes is also similar to the case of count based potentials. As before, we present only up to clique size of 3 × 3 as the learning code did not scale to larger clique sizes.

References

[1] C. Arora, S. Banerjee, P. Kalra, S. Maheshwari, Generalized flows for optimal inference in higher order MRF-map, IEEE Trans. Pattern Anal. Mach. Intell. (TPAMI) 37 (7) (2015) 1323–1335.
[2] D. Greig, B. Porteous, A.H. Seheult, Exact maximum a posteriori estimation for binary images, J. R. Stat. Soc. (Ser. B) 51 (2) (1989) 271–279.
[3] H. Ishikawa, Exact optimization for Markov random fields with convex priors, IEEE Trans. Pattern Anal. Mach. Intell. (TPAMI) 25 (10) (2003) 1333–1336.
[4] C. Arora, S. Maheshwari, Multi label generic cuts: Optimal inference in multi label multi clique MRF-map problems, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014.
[5] A. Globerson, T. Jaakkola, Fixing max-product: convergent message passing algorithms for map LP-relaxations., in: Advances in Neural Information Processing Systems (NIPS), 2007.
[6] N. Komodakis, N. Paragios, G. Tziritas, MRF optimization via dual decomposition: message-passing revisited, in: IEEE International Conference on Computer Vision (ICCV), 2007.
[7] D. Sontag, A. Globerson, T. Jaakkola, Introduction to dual decomposition for inference, in: Optimization for Machine Learning, MIT Press, 2011, pp. 219–254.
[8] D. Sontag, D.K. Choe, Y. Li, Efficiently searching for frustrated cycles in MAP inference, in: Conference on Uncertainty in Artificial Intelligence (UAI), 2012.
[9] Y. Boykov, V. Kolmogorov, An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision, IEEE Trans. Pattern Anal. Mach. Intell. (TPAMI) 26 (9) (2004) 1124–1137.
[10] A. Fix, T. Joachims, S.M. Park, R. Zabih, Structured learning of sum-of-submodular higher order energy functions, in: IEEE International Conference on Computer Vision (ICCV), 2013.
[11] A. Fix, C. Wang, R. Zabih, A primal-dual algorithm for higher-order multilabel Markov random fields, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014.
[12] V. Kolmogorov, Minimizing a sum of submodular functions, Discrete Appl. Math. 160 (15) (2012) 2246–2258.
[13] C. Wang, N. Komodakis, N. Paragios, Markov random field modeling, inference & learning in computer vision & image understanding: a survey, Comput. Vis. Image Understand. (CVIU) 117 (11) (2013) 1610–1627.
[14] D. Tarlow, I.E. Givoni, R.S. Zemel, Hop-map: efficient message passing with high order potentials, in: International Conference on Artificial Intelligence and Statistics (AISTATS), 2010.
[15] N. Komodakis, N. Paragios, Beyond pairwise energies: efficient optimization for higher-order MRFS, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2009.
[16] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, C. Rother, A comparative study of energy minimization methods for Markov random fields with smoothness-based priors, IEEE Trans. Pattern Anal. Mach. Intell. (TPAMI) 30 (6) (2008) 1068–1080.
[17] H. Ishikawa, Transformation of general binary MRF minimization to the first-order case, IEEE Trans. Pattern Anal. Mach. Intell. (TPAMI) 33 (6) (2011) 1234–1249.
[18] A. Fix, A. Gruber, E. Boros, R. Zabih, A graph cut algorithm for higher-order Markov random fields, in: IEEE International Conference on Computer Vision (ICCV), 2011.
[19] F. Kahl, P. Strandmark, Generalized roof duality, Discrete Appl. Math. 160 (16) (2012) 2419–2434.
[20] S. Iwata, J.B. Orlin, A simple combinatorial algorithm for submodular function minimization, in: ACM-SIAM Symposium on Discrete Algorithms (SODA), 2009.
[21] D. Sontag, T. Jaakkola, New outer bounds on the marginal polytope, in: Advances in Neural Information Processing Systems (NIPS), 2007.
[22] S. Riedel, Improving the accuracy and efficiency of MAP inference for Markov logic, in: Conference on Uncertainty in Artificial Intelligence (UAI), 2008.
[23] V. Kolmogorov, C. Rother, Minimizing nonsubmodular functions with graph cuts-a review, IEEE Trans. Pattern Anal. Mach. Intell. (TPAMI) 29 (7) (2007) 1274–1279.
[24] P. Kohli, P. Torr, Dynamic graph cuts for efficient inference in Markov random fields, IEEE Trans. Pattern Anal. Mach. Intell. (TPAMI) 29 (12) (2007) 2079–2088.
[25] Y. Boykov, O. Veksler, R. Zabih, Fast approximate energy minimization via graph cuts, IEEE Trans. Pattern Anal. Mach. Intell. (TPAMI) 23 (11) (2001) 1222–1239.
[26] C. Arora, S. Banerjee, P. Kalra, S. Maheshwari, An efficient graph cut algorithm for computer vision problems, in: European Conference on Computer Vision (ECCV), Springer, 2010.
[27] C. Papadimitriou, K. Steiglitz, Combinatorial Optimization, Prentice-Hall, Englewood Cliffs, NJ, 1982.
[28] H. Ishikawa, Higher-order clique reduction without auxiliary variables, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014.
[29] A database containing 1400 binary shape images, http://www.imageprocessingplace.com/root_files_V3/image_databases.htm (accessed: 2014-10-15).
[30] P. Stobbe, A. Krause, Efficient minimization of decomposable submodular functions, in: Advances in Neural Information Processing Systems(NIPS), 2010.
[31] SoS-IBFS implementation, http://www.cs.cornell.edu/~afix/Software/sum-of-submodular.tar.gz (accessed: 2015-04-30).
[32] C. Rother, P. Kohli, W. Feng, J. Jia, Minimizing sparse higher order energy functions of discrete variables, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2009.

[7] The learning code runs out of memory.