An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision

Yuri Boykov[1] and Vladimir Kolmogorov[2]

[1] Siemens Corporate Research, Imaging & Visualization, Princeton NJ 08540, USA
`yuri@csd.uwo.ca`
[2] Cornell University, Computer Science, Upson Hall, Ithaca NY 14853, USA
`vnk@cs.cornell.edu`

Abstract. After [10, 15, 12, 2, 4] minimum cut/maximum flow algorithms on graphs emerged as an increasingly useful tool for exact or approximate energy minimization in low-level vision. The combinatorial optimization literature provides many min-cut/max-flow algorithms with different polynomial time complexity. Their practical efficiency, however, has to date been studied mainly outside the scope of computer vision. The goal of this paper is to provide an experimental comparison of the efficiency of min-cut/max flow algorithms for energy minimization in vision. We compare the running times of several standard algorithms, as well as a new algorithm that we have recently developed. The algorithms we study include both Goldberg-style "push-relabel" methods and algorithms based on Ford-Fulkerson style augmenting paths. We benchmark these algorithms on a number of typical graphs in the contexts of image restoration, stereo, and interactive segmentation. In many cases our new algorithm works several times faster than any of the other methods making near real-time performance possible.

1 Introduction

Greig et. al. [10] were first to discover that powerful min-cut/max-flow algorithms from combinatorial optimization can be used to minimize certain important energy functions in vision. The energies addressed by Greig et. al. and by most later graph based methods (e.g. [15, 12, 2, 11, 4, 1, 18, 13, 16, 17, 3, 14]) can be represented as a posterior energy in MAP-MRF[1] framework:

$$E(L) = \sum_{p \in \mathcal{P}} D_p(L_p) + \sum_{(p,q) \in \mathcal{N}} V_{p,q}(L_p, L_q), \qquad (1)$$

where $L = \{L_p \,|\, p \in \mathcal{P}\}$ is a labeling of image $\mathcal{P}$, $D_p(\cdot)$ is a data penalty function, $V_{p,q}$ is an interaction potential, and $\mathcal{N}$ is a set of all pairs of neighboring pixels. Papers above show that, to date, graph based energy minimization methods provide arguably the most accurate solutions for the specified applications.

---

[1] MAP-MRF stands for Maximum *A Posterior* estimation of a Markov Random Field.

Greig et.al. constructed a two terminal graph such that the minimum cost cut of the graph gives a globally optimal binary labeling L in case of the Potts model of interaction in (1). Previously, exact minimization of energies like (1) was not possible and such energies were approached mainly with iterative algorithms like simulated annealing. In fact, Greig et.al. used their result to show that in practice simulated annealing reaches solutions very far from the global minimum even in very simple image restoration examples.

Unfortunately, the result of Greig et.al. remained unnoticed for almost 10 years mainly because the binary labeling limitation looked too restrictive. In the late 90's new computer vision techniques appeared that used min-cut/max-flow algorithms on graphs. [15] was the first to use these algorithms to compute multi-camera stereo. Later, [12, 2] showed that with the right edge weights on a similar to [15] graph one can minimize the energy in (1) for linear interaction penalties. The exact minimum could be computed when there are more than two labels. The results in [2, 4] showed that iteratively running min-cut/max-flow algorithms on appropriate graphs can be used to find provably good approximate solutions for even more general multi-label case when interaction penalties are *metrics*.

A growing number of publications in vision use graph based energy minimization techniques for applications like image segmentation [12, 18, 13, 3], restoration [10], stereo [15, 2, 11, 14], shape reconstruction [16], object recognition [1], augmented reality [17], and others. The graphs corresponding to these applications are usually huge 2D or 3D grids, and min-cut/max-flow algorithm efficiency is an issue that can not be ignored.

The goal of this paper is to compare experimentally the speed of several min-cut/max-flow algorithms on graphs typical for applications in vision. In Section 2 we provide basic facts about graphs, min-cut and max-flow problems, and some standard combinatorial optimization algorithms for them. Section 3 introduces a new min-cut/max-flow algorithm that we developed while working with graphs in vision. In Section 4 we tested our new algorithm and three standard min-cut/max-flow algorithms: H_PRF and Q_PRF versions of Goldberg-style "push-relabel" method [9, 5], and the Dinic algorithm [7]. We selected several examples in image restoration, stereo, and segmentation where different forms of energy (1) are minimized via graph structures originally described in [10, 12, 2, 4, 14, 3]. Such (or very similar) graphs are used in all computer vision papers known to us that use graph cut algorithms. In many interesting cases our new algorithm was significantly faster than the standard min-cut/max-flow techniques from combinatorial optimization. More detailed conclusions are presented in Section 5.

2 Background on Graphs

In this section we review some basic facts about graphs in the context of energy minimization methods in vision. A graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ consists of a set of nodes $\mathcal{V}$ and a set of directed edges $\mathcal{E}$ that connect them. Usually the nodes correspond to pixels, voxels, or other features. A graph normally contains some additional special nodes that are called terminals. In the context of vision, terminals cor-

respond to the set of labels that can be assigned to pixels. We will concentrate on the case of graphs with two terminals. Then the terminals are usually called the *source*, *s*, and the *sink*, *t*. In Figure 1(a) we show a simple example of a two terminal graph (due to Greig et. al. [10]) that can be used to minimize the Potts case of energy (1) on a 3×3 image with two labels. There is some variation in the structure of graphs used in other energy minimization methods in vision. However, most of them are based on regular 2D or 3D grid graphs as the one in Figure 1(a). This is a simple consequence of the fact that normal images (or volume data) in vision have grid-like structures.
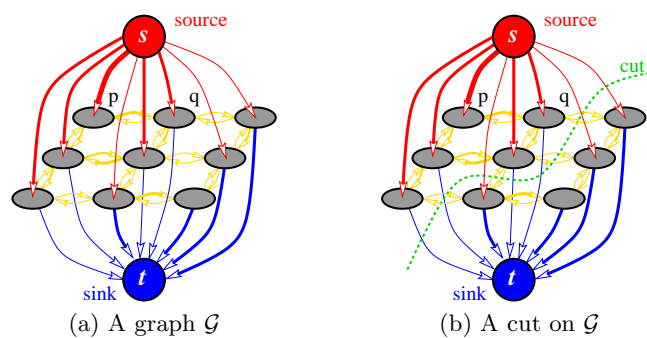

(a) A graph $\mathcal{G}$ (b) A cut on $\mathcal{G}$

Fig. 1. Example of a graph. Edge costs are reflected by their thickness. This graph construction was first used in Greig et. al. [10].

All edges in the graph are assigned some weight or cost. A cost of a directed edge (p, q) may differ from the cost of the reverse edge (q, p). Normally, there are two types of edges in the graph: n-links and t-links. N-links connect pairs of neighboring pixels or voxels. Thus, they represent a neighborhood system in the image. Cost of n-links corresponds to a penalty for discontinuity between the pixels. These costs are usually derived from the pixel interaction term $V_{p,q}$ in energy (1). T-links connect pixels with terminals (labels). The cost of a t-link connecting a pixel and a terminal corresponds to a penalty for assigning the corresponding label to the pixel. This cost is normally derived from the data term D_p in the energy function (1).

2.1   Min-Cut and Max-Flow Problems

An s/t cut (or just a *cut*) C on a graph with two terminals is a partitioning of the nodes in the graph into two disjoint subsets $\mathcal{S}$ and $\mathcal{T}$ such that the source s is in $\mathcal{S}$ and the sink t is in $\mathcal{T}$. Figure 1(b) shows one example of a cut. In combinatorial optimization the cost of a cut $C = \{\mathcal{S}, \mathcal{T}\}$ is defined as the sum of the costs of "boundary" edges (p, q) where $p \in \mathcal{S}$ and $q \in \mathcal{T}$. The *minimum cut* problem on a graph is to find a cut that has the minimum cost among all cuts.

One of the fundamental results in combinatorial optimization is that the minimum s/t cut problem can be solved by finding a *maximum flow* from the source s to the sink t. Loosely speaking, maximum flow is the maximum "amount of water" that can be sent from the source to the sink by interpreting graph edges as directed "pipes" with capacities equal to edge weights. The theorem of Ford and Fulkerson [8] states that a maximum flow from s to t saturates a set of edges in the graph dividing the nodes into two disjoint parts $\{\mathcal{S}, \mathcal{T}\}$ corresponding to a minimum cut. Thus, min-cut and max-flow problems are equivalent. In fact, the maximum flow value is equal to the cost of the minimum cut.

We can also intuitively show how min-cut (or max-flow) on a graph may help with energy minimization over image labelings. Consider an example in Figure 1. The graph corresponds to a 3×3 image. Any s/t cut partitions the nodes into disjoint groups each containing exactly one terminal. Therefore, any cut corresponds to some assignment of pixels (nodes) to labels (terminals). If edge weights are appropriately set based on parameters of an energy, a minimum cost cut will correspond to a labeling with the minimum value of this energy.[2]

2.2 Standard Algorithms in Combinatorial Optimization

An important fact in combinatorial optimization is that there are polynomial algorithms for min-cut/max-flow problems on graphs with two terminals. These algorithms can be divided into two main groups: Goldberg-style "push-relabel" methods and algorithms based on Ford-Fulkerson style augmenting paths.

Standard augmenting paths based algorithms, such as Dinic algorithm, work by pushing flow along non-saturated paths from the source to the sink until the maximum flow in the graph $\mathcal{G}$ is reached. A typical augmenting path algorithm stores information about the distribution of the current $s \to t$ flow f among the edges of $\mathcal{G}$ using a *residual graph* $\mathcal{G}_f$. The topology of $\mathcal{G}_f$ is identical to $\mathcal{G}$ but capacity of an edge in $\mathcal{G}_f$ reflects the residual capacity of the same edge in $\mathcal{G}$ given the amount of flow already in the edge. At the initialization there is no flow from the source to the sink (f=0) and edge capacities in the residual graph $\mathcal{G}_0$ are equal to the original capacities in $\mathcal{G}$. At each new iteration the algorithm finds the shortest $s \to t$ path along non-saturated edges of the residual graph. If a path is found then the algorithm *augments* it by pushing the maximum possible flow df that saturates at least one of the edges in the path. The residual capacities of edges in the path are reduced by df while the residual capacities of the reverse edges are increased by df. Each augmentation increases the total flow from the source to the sink $f = f + df$. The maximum flow is reached when any $s \to t$ path crosses at least one saturated edge in the residual graph $\mathcal{G}_f$.

Dinic algorithm uses breadth-first search to find the shortest paths from s to t on the residual graph $\mathcal{G}_f$. After all shortest paths of a fixed length k are saturated, the algorithm starts the breadth-first search for $s \to t$ paths of length

---

[2] Different graph based energy minimization methods may use different graph constructions, as well as, different rules for converting graph cuts into image labelings. Details for each method are described in the original publications.

$k + 1$ from scratch. Note that the use of shortest paths is an important factor that improves running time complexities for algorithms based on augmenting paths. The worst case running time complexity for Dinic algorithm is $O(mn^2)$ where n is the number of nodes and m is the number of edges in the graph.

Push-relabel algorithms use quite a different approach. They do not maintain a valid flow during the operation; each node may have a positive "flow excess", and the algorithm tries to push it to neighboring nodes. Push-relabel techniques are harder to describe in just a few sentences and we would rather refer the reader to our favorite text-book on basic graph theory and algorithms [6].

For our experimental tests on graph-based energy minimization methods in vision we selected the following standard algorithms.

DINIC: Algorithm of Dinic [7].
H_PRF: Push-Relabel algorithm [9] with the highest level selection rule.
Q_PRF: Push-Relabel algorithm [9] with the queue based selection rule.

Many previous experimental tests, including the results in [5], show that the last two algorithms work consistently better than a large number of other min-cut/max-flow algorithms of combinatorial optimization. The theoretical worst case complexities for these "push-relabel" algorithms are $O(n^3)$ for Q_PRF and $O(n^2\sqrt{m})$ for H_PRF.

3   New Min-Cut/Max-Flow Algorithm

In this section we present a new algorithm that we developed while working with graphs that are typical for energy minimization methods in computer vision. The algorithm presented here belongs to the group of algorithms based on augmenting paths. Similarly to DINIC it builds the search tree for finding augmenting paths but it reuses this tree and never starts building it from scratch. The drawback of our approach is that the augmenting paths found are not necessarily shortest augmenting path; thus the time complexity of the shortest augmenting path is no longer valid. The trivial upper bound on the number of augmentations for our algorithm is the cost of the minimum cut $|C|$, which results in the worst case complexity $O(mn^2|C|)$. Theoretically speaking, this is worse than complexities of the standard algorithms discussed in Section 2.2. However, experimental comparison in Section 4 shows that on typical problem instances in vision our algorithm significantly outperforms standard algorithms.

3.1   Algorithm's Overview

We maintain a search tree S with the source as a root where all edges from each parent node to its children are non-saturated. The nodes that are not in S are called "free". The set of free nodes is denoted T. The nodes in the search tree S are divided into "active" and "passive". The active nodes may "grow", that is, they may acquire new children from a set of free nodes. The passive nodes are

guaranteed to have no free neighbors connected through non-saturated edges. Thus, the passive nodes can not grow.

The algorithm iteratively repeats the following three stages:

- "growth" stage: the search tree grows until the sink is found
- "augmentation" stage: the path found is augmented, the search tree is broken into a forest.
- "adoption" stage: the forest is transformed back into a tree.

At the growth stage the search tree expands. The active nodes acquire new children from a set of free nodes. The newly acquired nodes become active members of the search tree S. As soon as all neighbors of a given active node are explored the active node becomes passive. The growth stage terminates when the sink is encountered and, thus, a path from the source to the sink is found.

The augmentation stage augments the path found in the growth stage. Since we push through the largest flow possible some edges in the path become saturated. Thus, some of the nodes in the tree become "orphans", that is, the edges linking them to their parents are no longer valid (they are saturated). In fact, the augmentation phase splits the search tree S into a forest. The source is still a root of one of the trees in the forest and the orphans form roots of other trees.

The goal of the adoption stage is to restore a single search tree structure with a root in the source. At this stage we try to find a new valid parent for each orphan. If there is no such parent we remove the orphan from S and make it a free node. We also declare all its former children orphans. The stage terminates when no orphans are left and, thus, the search tree structure of S is restored. Since some orphan nodes in S may become free the adoption stage results in contraction of the set S.

After the adoption stage is completed the algorithm returns to the growth stage. The algorithm terminates when the search tree can not grow (all active nodes checked their neighbors and became passive) while the sink is not found.

3.2 Details of Implementation

Assume that we are given a directed graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$. As for any augmenting path algorithm, we will maintain a flow f and the residual graph G_f (see Section 2.2). For each node p we will store its parent as $PARENT(p)$. Roots of the forest (the source and the orphans) as well as all free nodes have no parents, t.e. $PARENT(p) = \emptyset$. We will also keep the lists of all active nodes, A, and all orphans, O. The general structure of the algorithm is:

```
initialize: S = A = {s}, T = V − {s}, O = ∅
while true
            grow S to find an augmenting path P from s to t
            if P = ∅ terminate
            augment on P
            adopt orphans
end while
```

The details of the *growth*, *augmentation*, and *adoption* stages are described below.

Growth stage: At this stage active nodes acquire new children from a set of free nodes.

```
if  t ∈ S  return  P = PATH_{s→t}
while  A ≠ ∅
      pick an active node  p ∈ A
      for every non-saturated edge  (p,q)
          if  q ∈ T  add  q  to the search tree as an active node:
              S := S ∪ {q},  A := A ∪ {q},  PARENT(q) := p
          if  q = t  return  P = PATH_{s→t}
      end for
      remove  p  from  A
end while
return  P = ∅
```

Augmentation stage: The input for this stage is a path P from s to t. Note that the orphan set is empty in the beginning of the stage, but there will be some orphans in the end since at least one edge in P becomes saturated.

```
find the bottleneck capacity  Δ  on  P
update the residual graph by pushing flow  Δ  through  P
for each edge  (p,q)  in  P  that becomes saturated
      set  PARENT(q) := ∅
      add  q  to  O
end for
```

Adoption stage: During this stage all nodes in O are processed until O becomes empty. The node being processed tries to find a new parent in S; in case of success it remains in S but with a new parent, otherwise it is removed from S to the set of free nodes T and all its children are added to O.

```
while  O ≠ ∅
      pick a node  p ∈ O
      remove  p  from  O
      process  p
end while
```

The operation "`process` p" consists of the following steps. First we are trying to find a new parent for p. For each non-saturated edge (q,p) entering p we check whether q is a valid parent. Two conditions should hold for q:

- q should be in S
- the "origin" of q should be the source

Note that it is necessary to check the second condition because some of the nodes in S originate from orphans.

If a new parent q is found, then p remains in S with q as its parent. The active (or passive) status of p in S remains unchanged. If p does not find a valid parent in S then the following three operations are performed:

- p is removed from S (and A) and becomes a free node in T
- for all children q of p we set $PARENT(q) = \emptyset$ and add them to the set of orphans O
- all "potential" parents of p (nodes q in S such that the edge (q,p) is not saturated) are added to the active set A

The last operation is necessary to make sure that no passive node in S connects to a free neighbor through a non-saturated edge. Only active nodes are allowed to have such free neighbors. Suppose that an orphan p becomes free. Without the last operation, the passive neighbors of p in S connected to p via non-saturated edges would remain passive while they should not. At that moment these neighbors did not qualify as valid parents for p because they originated from other orphans and not from the source. After the search tree is fixed one of such neighbors may potentially become a new parent of p.

3.3 Correctness proof

Let's introduce some invariants which are maintained during the execution of the algorithm.

I1 *S is a forest with roots at either the source or orphans.*
I2 *Edges from a parent to children in the search forest have nonzero residual capacities.*
I3 *There are no orphans during the growth stage.*
I4 *For passive nodes p in S the following property should be true: for all non-saturated edges (p,q) the node q must belong to S.*

These invariants are clearly true at the initialization of the algorithm. It is easy to see these invariants directly follow from the construction of the algorithm.

Let's show that all stages terminate. The growth stage terminates because the number of nodes is finite. The same argument applies to the augmentation stage. Now we prove that the adoption stage is also finite. Note that after a node p in O has been processed it can not become an orphan again during the same adoption stage (it will imply that the adoption stage terminates after processing at most n nodes). Indeed, if p is moved from S to T then this holds since free nodes in T are not involved at the adoption stage. Suppose p found a new parent q and remained in S. The new parent q must originate from the source. Thus, the source is the new origin of p as well. By construction, only descendants of orphans may become orphans during the adoption stage. Therefore, p can not become an orphan again at the same adoption stage.

The algorithm terminates if the number of cycles (augmentations) is finite. Since the algorithm is not a shortest path algorithm the polynomial bound for the number of augmentations does not seem to be valid. We know only a trivial bound given by a minimum cut cost that works if all edge weights are integers.

It remains to show that when the algorithm terminates it generates the maximum flow. In fact, the search tree S and the set of free nodes T at the end of

the algorithm give a minimum s/t-cut. Suppose the algorithm has terminated. It could only have happened in the growth stage when no active nodes were left and $t \notin S$. S and T are disjoint sets such that $S \cup T = \mathcal{V}$, $s \in S$, and $t \in T$. Suppose that the current residual graph contains a non-saturated path from the source to the sink that can be used to increase the flow. Then there is a non-saturated edge (p, q) going from a node $p \in S$ to another node $q \in T$. Since no active nodes are left then p is passive. Hence, the invariant I4 does not hold for p and we get a contradiction.

4 Experimental Tests on Applications in Vision

In this section we experimentally test min-cut/max-flow algorithms for three different applications in computer vision: image restoration (Section 4.1), stereo (Section 4.2), and object segmentation (Section 4.3). We chose formulations where certain appropriate versions of energy (1) can be minimized via graph cuts. The corresponding graph structures were previously described by [10, 12, 2, 4, 14, 3] in detail. These (or very similar) structures are used in all computer vision applications with graph cuts (that we are aware of) to date.

Note that we could not test all known min-cut/max-flow algorithms. We compare our new algorithm presented in Section 3 and standard algorithms of combinatorial optimization introduced in Section 2.2: DINIC, H_PRF, and Q_PRF. Many experimental tests, including the results in [5], show that the last two algorithms work consistently better than a large number of other min-cut/max-flow algorithms of combinatorial optimization. For DINIC, H_PRF, and Q_PRF we took the implementations written by Cherkassky and Goldberg [5] and modified them to our graph representation. Both H_PRF and Q_PRF use global and gap relabeling heuristics. Our algorithm also leaves some choice in implementing certain functions. We found that the order of processing active nodes and orphans may have a significant effect on the running time. We made a tuning and used it in all experiments.

4.1 Image Restoration

Here we consider two examples of energy (1) with the Potts and linear models of interaction. Graph based methods for minimizing Potts energy were used in many different applications including segmentation [13], stereo [2, 4], object recognition [1], shape reconstruction [16], and augmented reality [17]. Linear interaction energy was used for stereo [15] and segmentation [12]. The structures of the corresponding graphs are identical in all applications using the same type of energy. We chose the context of image restoration mainly for its simplicity.

The Potts energy that we use for image restoration is

$$E(I) = \sum_{p \in \mathcal{P}} ||I_p - I_p^o|| + \sum_{(p,q) \in \mathcal{N}} K_{(p,q)} \cdot T(I_p \neq I_q) \tag{2}$$

(a) *Diamond* restoration (b) Original *Bell Quad* (c) "Restored" *Bell Quad*

method	input		input: *Diamond*			input: *BellQuad*		
	Diamond	*Bell Quad*	$\|\mathcal{L}\|$=27	$\|\mathcal{L}\|$=54	$\|\mathcal{L}\|$=108	$\|\mathcal{L}\|$=32	$\|\mathcal{L}\|$=64	$\|\mathcal{L}\|$=128
DINIC	21	160	24	61	177	24	70	144
H_PRF	10	22	10	22	53	16	50	125
Q_PRF	10	23	7	20	54	9	19	59
Our	6	14	5	16	65	8	27	122

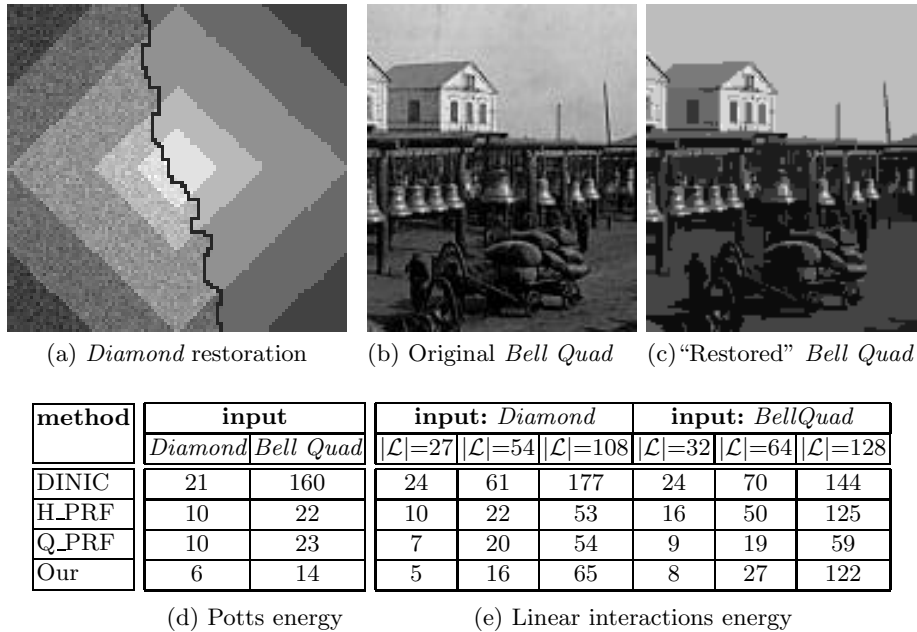(d) Potts energy   (e) Linear interactions energy

Fig. 2. Image Restoration Experiments

where $I = \{I_p \mid p \in \mathcal{P}\}$ is a vector of unknown "true" intensities of pixels on the image $\mathcal{P}$ and $I^o = \{I_p^o \mid p \in \mathcal{P}\}$ are intensities observed in the original image corrupted by noise. The Potts interactions are specified by penalties $K_{(p,q)}$ for intensity discontinuities between pairs of neighboring pixels. Function $T(\cdot)$ is 1 if the condition inside parenthesis is true and 0 otherwise. In the case of two labels the Potts energy can be minimized exactly using the graph cut method of Greig et. al. [10]. We consider image restoration with multiple labels where the problem becomes NP hard. We use an iterative graph based method in [4] which is guaranteed to find a solution within a factor of two from the global minimum of the Potts energy. At each iteration [4] computes a minimum cost cut for a certain generalization of the graph introduced in [10].

Our image restoration experiments with the Potts energy are presented in Figure 2(a-c). The sizes of our test images are 100×100 (Diamond) and 112×136 (Bell Quad). The number of allowed labels is 215 and 256, correspondingly. The running times (in seconds, 333MHz Pentium III) for the Potts energy minimization tests are given in Figure 2(d). These running times represent the first cycle of iterations (see [4] for more details).

We also consider image restoration with "linear" interactions energy:

$$E(I) = \sum_{p \in \mathcal{P}} ||I_p - I_p^o|| + \sum_{(p,q) \in \mathcal{N}} A_{(p,q)} \cdot |I_p - I_q| \tag{3}$$

where constants $A_{(p,q)}$ describe the relative importance of interactions between neighboring pixels p and q. If the set of labels is finite and ordered then this energy can be minimized exactly using either of the two almost identical graph-based methods developed in [12, 2]. In fact, both of them use graphs very similar to the one introduced by [15] in the context of multi-camera stereo. These methods build graphs by consecutively connecting multiple layers of image-grids. Each layer corresponds to one label. The structure of the graphs for linear interactions energy has one important distinction from the graphs that are currently used to minimize other types of energies; the two terminals are connected only to the first and the last layers of the graph. This distinction becomes more pronounced when the number of labels (layers) is large. Note that allocating computer memory for such multi-layered graphs can be problematic even for 2D images.

The table in Figure 2(e) shows how long it took each min-cut/max-flow algorithm to compute the exact minimum of the linear interactions energy above. We used the same *Diamond* and *Bell Quad* images as in the Potts energy tests. In the tests presented in (e) we varied the number of labels (layers) $|\mathcal{L}|$. The experiments show that our algorithm is the fastest when the number of labels is relatively small (less than 50) while Q_PRF wins for larger number of labels. Note that the number of labels affects the structure of the graphs in [15, 12, 2]. In the Potts energy minimization method in [4] the number of labels changes the number of iterations in each cycle but has no effect on the graph structures.

4.2 Stereo with Occlusions

Here we describe our tests on examples in stereo. We consider a recent formulation [14] that takes occlusions into consideration. The problem is formulated as a labeling problem. We want to assign a binary label (0 or 1) to each pair $\langle p, q \rangle$ where p is a pixel in the left image and q is a pixel in the right image that can potentially correspond to p. The set of pairs with the label 1 describes the correspondence between the images. The energy of configuration f is given by

$$E(f) = \sum_{f_{\langle p,q \rangle}=1} D_{\langle p,q \rangle} + \sum_{p \in \mathcal{P}} C_p \cdot T(p \text{ is occluded in the configuration } f)$$

$$+ \sum_{\{\langle p,q \rangle, \langle p',q' \rangle\} \in \mathcal{N}} K_{\{\langle p,q \rangle, \langle p',q' \rangle\}} \cdot T(f_{\langle p,q \rangle} \neq f_{\langle p',q' \rangle})$$

The first term is the data term, the second is the occlusion penalty, and the third is the smoothness term. $\mathcal{P}$ is the set of pixels in both images, and $\mathcal{N}$ is the neighboring system consisting of tuples of neighboring pairs $\{\langle p,q \rangle, \langle p',q' \rangle\}$ having the same disparity (parallel pairs). [14] gives an approximate algorithm minimizing this energy among all feasible configurations f. In contrast to other energy minimization methods, nodes of the graph constructed in [14] represent *pairs* rather than pixels or voxels.

The tests were done for three stereo examples shown in Figure 3. We used the *Head* pair from the University of Tsukuba, and the well-known *Tree* pair from
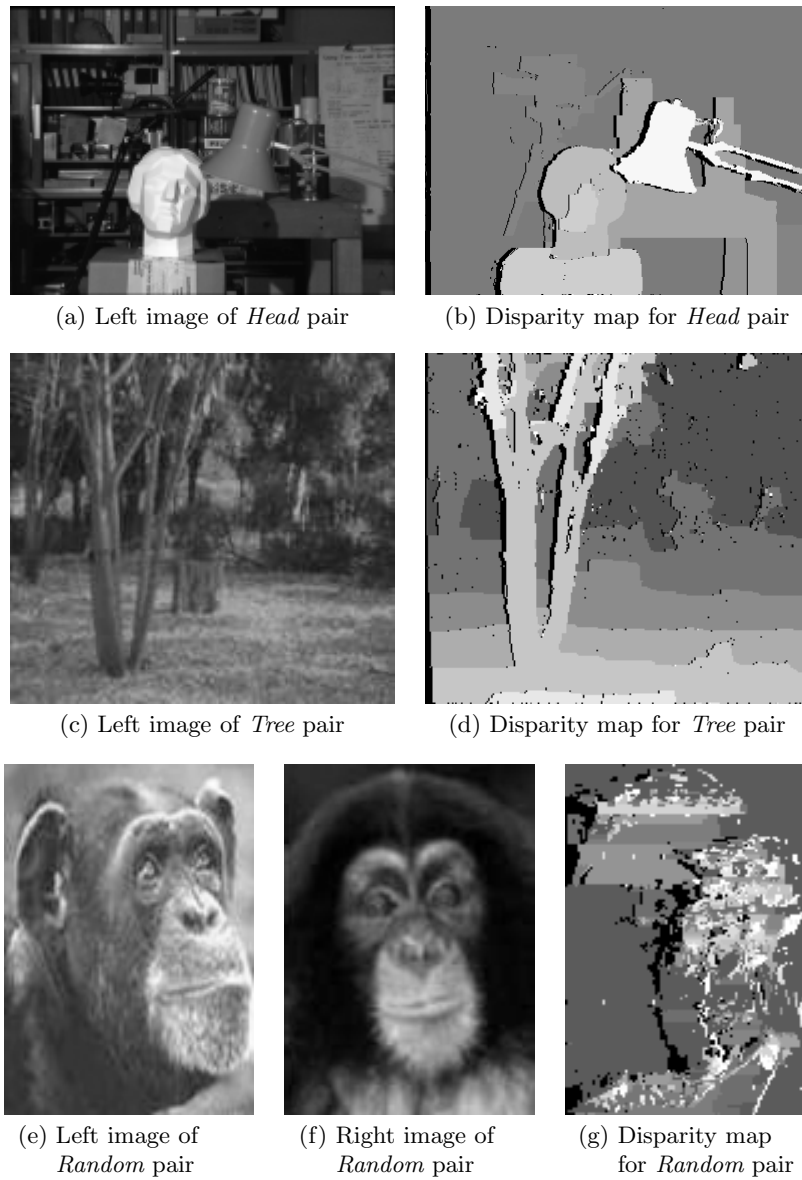
(a) Left image of *Head* pair

(b) Disparity map for *Head* pair

(c) Left image of *Tree* pair

(d) Disparity map for *Tree* pair

(e) Left image of
Random pair

(f) Right image of
Random pair

(g) Disparity map
for *Random* pair

Fig. 3. Stereo Experiments. The sizes of images are 384×288 in (a), 256×233 in (c), and 100×140 in (e,f). The results in (b,d,g) show occluded pixels in black color.

SRI. To diversify our tests we compared the speed of algorithms on a *Random* pair where the left and the right images did not correspond to each other.

Running times for stereo examples in Figure 3 are shown in seconds (450MHz UltraSPARC II Processor) in the table below. The times are for the first cycle of the algorithm, which is where most of the work is done.

method	input		
	Head pair	*Tree* pair	*Random* pair
DINIC	365.4	39.4	32.6
H_PRF	109.8	20.1	16.0
Q_PRF	56.0	13.4	9.1
Our	17.0	4.0	7.2

4.3 Interactive Object Segmentation

In this section we describe experimental tests that compare min-cut/max-flow algorithms on *Interactive Graph Cuts* segmentation technique in [3]. The method in [3] allows for the segmentation of an object of interest in N-D images/volumes. This technique generalizes the MAP-MRF method of Greig at. al. [10] by incorporating additional hard constraints into the minimization of the Potts energy

$$E(L) = \sum_{p \in \mathcal{P}} D_p(L_p) + \sum_{(p,q) \in \mathcal{N}} K_{(p,q)} \cdot T(L_p \neq L_q)$$
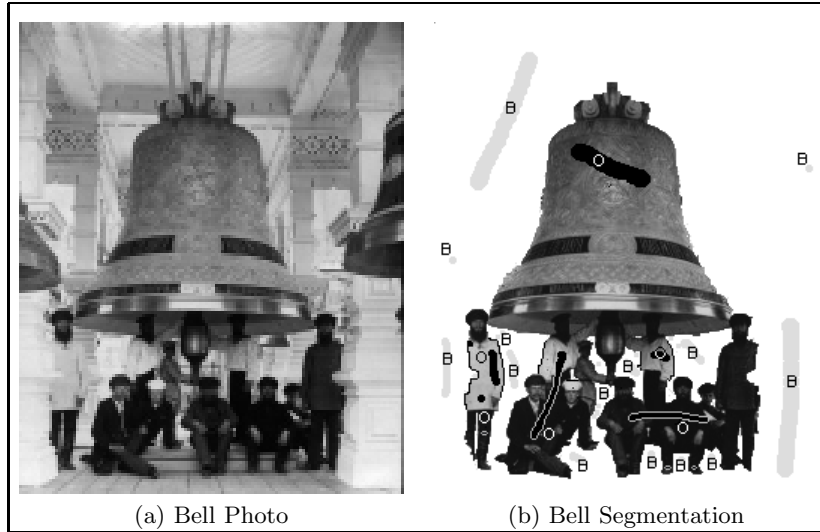
over binary (object/background) labelings of image. The hard constrains come from a user placing some object and background seeds. The technique computes binary segmentation of N-dimensional image with globally optimal regional and boundary properties among all segmentations that satisfy the hard constraints (seeds). The details of the corresponding graph construction are given in [3].

We tested min-cut/max-flow algorithms on 2D and 3D segmentation examples illustrated in Figure 4. We present the original data and the segmentation results corresponding to certain sets of seeds. Note that the user places seeds interactively. New seeds can be added to correct segmentation imperfections. The technique in [3] efficiently recomputes the optimal solution starting at the previous segmentation result.

Figure 4(a-b) shows photo-editing experiment on a picture (200x300 pixels) with a group of people around a bell. Other segmentation examples in (c-h) are for 2D and 3D medical data. The cardiac MR data in (c-d) was tested in both 2D (256x256 pixels) and 3D (256x256x13 voxels) cases. In our 3D experiment the seeds were placed in only one slice in the middle of the volume. This was enough to segment the whole volume "correctly". The tests with lung CT data (e-f) were also made in both 2D (512x512 pixels) and 3D (512x512x5 voxels) cases. In (g-h) we tested the algorithms on 2D liver MR data (512x256 pixels).

The table below compares the running times (in seconds, 600MHz Pentium III processor) of selected min-cut/max-flow algorithms for the segmentation examples described. Note that these times include only the min-cut/max-flow com-
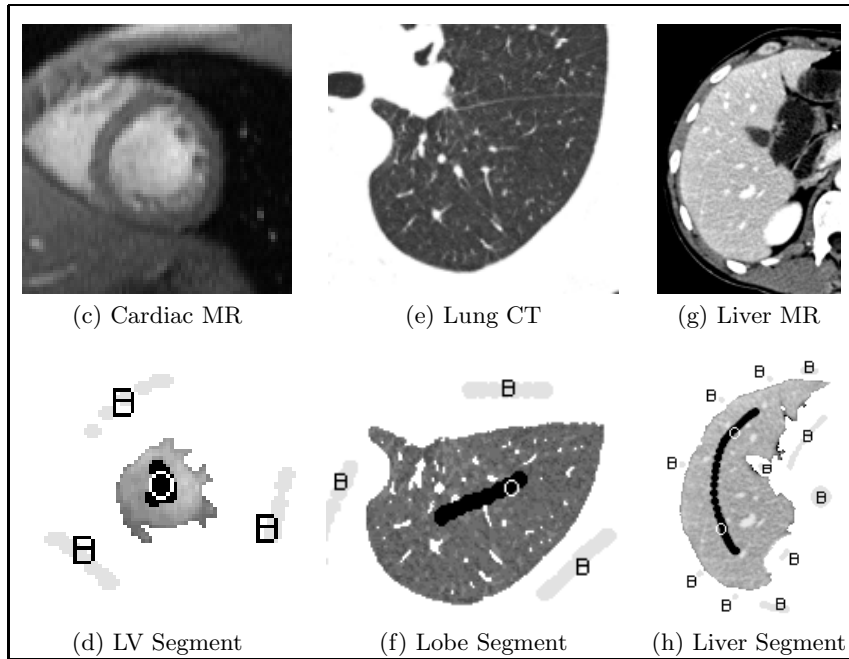
Photo Editing

(a) Bell Photo (b) Bell Segmentation

Medical Data

(c) Cardiac MR (e) Lung CT (g) Liver MR

(d) LV Segment (f) Lobe Segment (h) Liver Segment

Fig. 4. Segmentation Experiments

putation[3]. The tests on 3D data are marked by "3D". To diversify our tests we also made a few experiments where inconsistent seeds were placed at random places in the image. The corresponding columns in the table are marked as "random". Meaningless segmentations from these tests are not shown in Figure 4.

method	input						
	Liver	Bell	Lung	Heart(3D)	Lung(3D)	Bell(random)	Lung(random)
DINIC	26	7	5	—	—	8	16
H_PRF	3.5	2	2.5	—	—	4	3
Q_PRF	2.5	1	0.5	7	68	1.5	1.5
Our	0.26	0.26	0.16	2	20	1	2.5

Note that 3D segmentation required memory efficient implementations of the graph cut algorithms. We made such implementations only for our new algorithm and for Q_PRF (which outperformed H_PRF and DINIC in most other experiments). H_PRF and DINIC were not tested in 3D segmentation examples.

5   Conclusions

We tested a reasonable sample of typical vision graphs. In most examples our new min-cut/max-flow algorithm worked 2-10 times faster than any of the other methods, including the push-relabel and Dinic algorithms (which are known to outperform other min-cut/max-flow techniques). In some cases the new algorithm made possible near real-time performance of the corresponding applications. One noticeable exception was the energy with linear interactions (3). If the number of labels for (3) was relatively small (< 50) then our algorithm was only marginally the best, while Q_PRF was significantly faster for larger number of labels. We also found that our algorithm's performance was roughly the same as Q_PRF in unrealistic examples with "random" inputs.

Our results also suggest that graphs in vision are a very specific application for min-cut/max-flow algorithms. In fact, Q_PRF outperformed H_PRF in most of our tests despite the fact that H_PRF is generally regarded as the fastest algorithm in combinatorial optimization community. Additional experiments showed that our algorithm was several times slower than H_PRF on standard (outside computer vision) graphs that are used for tests in combinatorial optimization.

Acknowledgements

---

[3] The time it takes the user to place the seeds varies and may depend on image quality, object of interest, and the level of desired details. For the experiments in Figure 4 all seeds were placed within 10 to 40 seconds.

References

1. Y. Boykov and D. Huttenlocher. A new bayesian framework for object recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 517–523, 1999.

2. Y. Boykov, O. Veksler, and R. Zabih. Markov random fields with efficient approximations. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 648–655, 1998.

3. Yuri Boykov and Marie-Pierre Jolly. *Interactive graph cuts* for optimal boundary & region segmentation of objects in N-D images. In *International Conference on Computer Vision*, July 2001.

4. Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. In *International Conference on Computer Vision*, volume I, pages 377–384, 1999.

5. B. V. Cherkassky and A. V. Goldberg. On implementing push-relabel method for the maximum flow problem. *Algorithmica*, 19:390–410, 1997.

6. William J. Cook, William H. Cunningham, William R. Pulleyblank, and Alexander Schrijver. *Combinatorial Optimization*. John Wiley & Sons, 1998.

7. E. A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Math. Dokl.*, 11:1277–1280, 1970.

8. L. Ford and D. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

9. A. Goldberg and R. Tarjan. A new approach to the maximum flow problem. *Journal of the Association for Computing Machinery*, 35(4):921–940, October 1988.

10. D. Greig, B. Porteous, and A. Seheult. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society, Series B*, 51(2):271–279, 1989.

11. H. Ishikawa and D. Geiger. Occlusions, discontinuities, and epipolar lines in stereo. In *5th European Conference on Computer Vision*, pages 232–248, 1998.

12. H. Ishikawa and D. Geiger. Segmentation by grouping junctions. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 125–131, 1998.

13. Junmo Kim, John W. Fisher III, Andy Tsai, Cindy Wible, Alan S. Willsky, and William M. Wells III. Incorporating spatial priors into an information theoretic approach for fMRI data analysis. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 62–71, 2000.

14. Vladimir Kolmogorov and Ramin Zabih. Computing visual correspondence with occlusions via graph cuts. In *International Conference on Computer Vision*, July 2001.

15. Sebastien Roy and Ingemar Cox. A maximum-flow formulation of the n-camera stereo correspondence problem. In *IEEE Proc. of Int. Conference on Computer Vision*, pages 492–499, 1998.

16. Dan Snow, Paul Viola, and Ramin Zabih. Exact voxel occupancy with graph cuts. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 345–352, 2000.

17. B. Thirion, B. Bascle, V. Ramesh, and N. Navab. Fusion of color, shading and boundary information for factory pipe segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 349–356, 2000.

18. Olga Veksler. Image segmentation by nested cuts. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 339–344, 2000.