PAMI submission: TPAMI-0120-0603, first submitted in May, 2002

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

Yuri Boykov and Vladimir Kolmogorov*

Abstract

After [13, 22, 14, 15, 3, 6] 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 segmentation. In many cases our new algorithm works several times faster than any of the other methods making near real-time performance possible. An implementation of our max-flow/min-cut algorithm is available upon request for research purposes.

Index Terms — Energy minimization, graph algorithms, minimum cut, maximum flow, image restoration, segmentation, stereo, multi-camera scene reconstruction.

*Yuri Boykov is with Siemens Research, Princeton, NJ, yuri@scr.siemens.com. Vladimir Kolmogorov is with the Computer Science Department, Cornell University, Ithaca, NY 14853, vnk@cs.cornell.edu

1 Introduction

Greig et al. [13] 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. [22, 15, 3, 14, 6, 2, 27, 18, 24, 26, 4, 20, 21, 7]) can be represented as a posterior energy in MAP-MRF[1] framework:

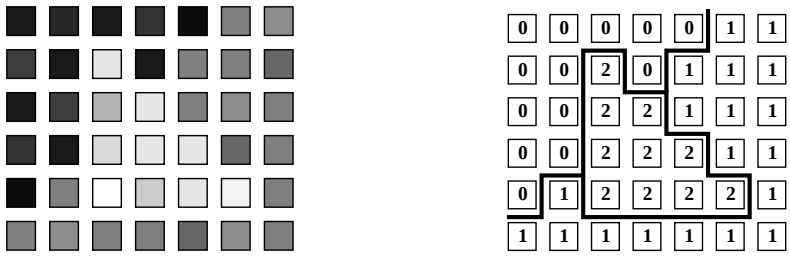
$$E(L) = \sum_{p \in \mathcal{P}} D_p(L_p) + \sum_{(p,q) \in \mathcal{N}} V_{p,q}(L_p, L_q), \tag{1}$$

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. An example of image labeling is shown in Figure 1. Typically, data penalties $D_p(\cdot)$ indicate individual label-preferences of pixels based on observed intensities and pre-specified likelihood function. Interaction potentials $V_{p,q}$ encourage spatial coherence by penalizing discontinuities between neighboring pixels. Papers above show that, to date, graph-based energy minimization methods provide arguably some of the most accurate solutions for the specified applications. For example, consider two recent evaluations of stereo algorithms using real imagery with dense ground truth [23, 25].

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 a very simple example of binary image restoration.

Unfortunately, the graph cut technique in Greig et al. remained unnoticed for almost 10 years mainly because binary image restoration looked very limited as an application. In the late 90's new computer vision techniques appeared that figured how to use min-cut/max-flow algorithms on graphs for more interesting non-binary problems. [22] was the first to use these algorithms to compute multi-camera stereo. Later, [15, 3] showed that with the right edge weights on a graph similar to that used in [22] one can minimize a non-binary case of (1) with linear interaction

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



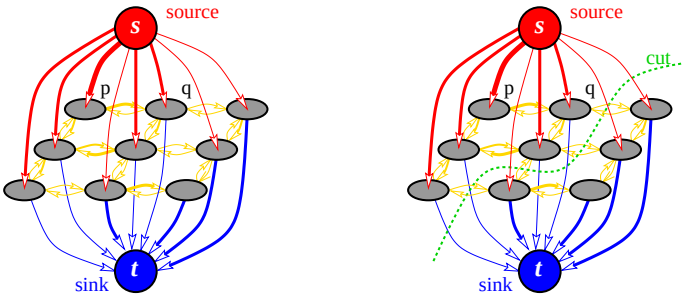
(a) An image (b) A labeling

Figure 1: An example of image labeling. An image in (a) is a set of pixels $\mathcal{P}$ with observed intensities I_p for each $p \in \mathcal{P}$. A labeling L shown in (b) assigns some label $L_p \in \{0, 1, 2\}$ to each pixel $p \in \mathcal{P}$. Such labels can represent depth (in stereo), object index (in segmentation), original intensity (in image restoration), or other pixel properties. Normally, graph-based methods assume that a set of feasible labels at each pixel is finite. Thick lines in (b) show labeling discontinuities between neighboring pixels.

penalties. This graph construction was further generalized to handle convex cliques in [16]. The results in [3, 6] showed that iteratively running min-cut/max-flow algorithms on appropriate graphs can be used to find provably good approximate solutions for the general multi-label case in which the interaction penalty is a *metric*.

A growing number of publications in vision use graph-based energy minimization techniques for applications like image segmentation [15, 27, 18, 4], restoration [13], stereo [22, 3, 14, 20, 21, 7], shape reconstruction [24], object recognition [2], augmented reality [26], 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 cannot be ignored.

The goal of this paper is to compare experimentally the running time 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: the H_PRF and Q_PRF versions of the Goldberg-style "push-relabel" method [12, 8], and the Dinic algorithm [10]. We selected several examples in image



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

Figure 2: Example of a directed capacitated graph. Edge costs are reflected by their thickness. A similar graph-cut construction was first used in vision by Greig et al. [13] for binary image restoration.

restoration, stereo, and segmentation where different forms of energy (1) are minimized via graph structures originally described in [13, 15, 3, 6, 20, 21, 4]. 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 directed weighted (capacitated) 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 correspond 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 2(a) we show a simple example of a two terminal graph (due to Greig et al. [13]) 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 2(a). This is a simple consequence of the fact that normally graph nodes represent regular image pixels or voxels.

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). In fact, ability to assign different edge weights for (p, q) and (q, p) is important for many graph-based applications in vision. 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 (1).

2.1 Min-Cut and Max-Flow Problems

An s/t 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 $\mathcal{T}$. For simplicity, throughout this paper we refer to s/t cuts as just *cuts*. Figure 2(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 $q \in \mathcal{T}$. Note that cut cost is "directed" as it sums up weights of directed edges specifically from $\mathcal{S}$ to $\mathcal{T}$. 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 [11] 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 (see also Fig. 4). 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 2. 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 directed weighted graphs with two terminals. These algorithms can be divided into two main groups: Goldberg-Tarjan style "push-relabel" methods [12] and algorithms based on Ford-Fulkerson style augmenting paths [11]. There are also some randomized algorithms (e.g. [17]) for min-cut/max-flow problems but they are suitable only for dense undirected graphs which are not relevant for energy minimization in vision.

Standard augmenting paths based algorithms, such as Dinic algorithm [10], 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 $s \rightarrow t$ 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 $s \rightarrow t$ 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 $s \rightarrow t$ 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

[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.

breadth-first search for $s \rightarrow t$ paths of length $k+1$ 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 [12] use quite a different approach. They do not maintain a valid flow during the operation; there are "active" nodes that have a positive "flow excess". Instead, the algorithms maintain a labeling of nodes giving a low bound estimate on the distance to the sink along non-saturated edges. The algorithms attempt to "push" excess flows towards nodes with smaller estimated distance to the sink. Typically, the "push" operation is applied to active nodes with the largest distance (label) or based on FIFO selection strategy. The distances (labels) progressively increase as edges are saturated by push operations. Undeliverable flows are eventually drained back to the source. We recommend our favorite text-book on basic graph theory and algorithms [9] for more details on push-relabel and augmenting path methods.

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 [10] it builds search trees for detecting augmenting paths. In fact, we build two search trees, one from the source and the other from the sink[3]. The other difference is that we reuse these trees and never start building them 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]Note that in the earlier publication [5] we used a single tree rooted at the source that searched for the sink. The two-trees version presentred here treats the terminals simmetrically. Experimentally, the new algorithm consistently outpermes the one in [5].

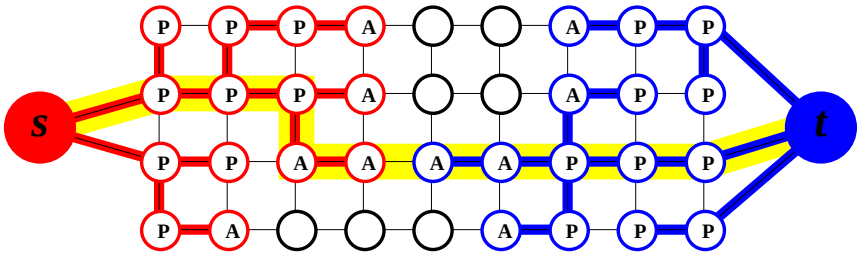


Figure 3: Example of the search trees S (red nodes) and T (blue nodes) at the end of the growth stage when a path (yellow line) from the source s to the sink t is found. Active and passive nodes are labeled by letters `A` and `P`, correspondingly. Free nodes appear in black.

3.1 Algorithm's Overview

Figure 3 illustrates our basic terminology. We maintain two non-overlapping search trees S and T with roots at the source s and the sink t, correspondingly. In tree S all edges from each parent node to its children are non-saturated, while in tree T edges from children to their parents are non-saturated. The nodes that are not in S or T are called "free". We have

$$S \subset \mathcal{V}, \quad s \in S, \quad T \subset \mathcal{V}, \quad t \in T, \quad S \cap T = \emptyset.$$

The nodes in the search trees S and T can be either "active" or "passive". The active nodes represent the outer border in each tree while the passive nodes are internal. The point is that active nodes allow trees to "grow" by acquiring new children (along non-saturated edges) from a set of free nodes. The passive nodes can not grow as they are completely blocked by other nodes from the same tree. It is also important that active nodes may come in contact with the nodes from the other tree. An augmenting path is found as soon as an active node in one of the trees detects a neighboring node that belongs to the other tree.

The algorithm iteratively repeats the following three stages:

- "growth" stage: search trees S and T grow until they touch giving an $s \to t$ path
- "augmentation" stage: the found path is augmented, search tree(s) brake into forest(s)
- "adoption" stage: trees S and T are restored.

At the growth stage the search trees expand. The active nodes explore adjacent non-saturated edges and acquire new children from a set of free nodes. The newly acquired nodes become active members of the corresponding search trees. As soon as all neighbors of a given active node are explored the active node becomes passive. The growth stage terminates if an active node encounters a neighboring node that belongs to the opposite tree. In this case we detect a path from the source to the sink, as shown in Figure 3.

The augmentation stage augments the path found at the growth stage. Since we push through the largest flow possible some edge(s) in the path become saturated. Thus, some of the nodes in the trees S and T may become "orphans", that is, the edges linking them to their parents are no longer valid (they are saturated). In fact, the augmentation phase may split the search trees S and T into forests. The source s and the sink t are still roots of two of the trees while orphans form roots of all other trees.

The goal of the adoption stage is to restore single-tree structure of sets S and T with roots in the source and the sink. At this stage we try to find a new valid parent for each orphan. A new parent should belong to the same set, S or T, as the orphan. A parent should also be connected through a non-saturated edge. If there is no qualifying parent we remove the orphan from S or T and make it a free node. We also declare all its former children as orphans. The stage terminates when no orphans are left and, thus, the search tree structures of S and T are restored. Since some orphan nodes in S and T may become free the adoption stage results in contraction of these sets.

After the adoption stage is completed the algorithm returns to the growth stage. The algorithm terminates when the search trees S and T can not grow (no active nodes) and the trees are separated by saturated edges. This implies that a maximum flow is achieved. The corresponding minimum cut can be determined by $\mathcal{S} = S$ and $\mathcal{T} = T$.

3.2 Details of Implementation

Assume that we have 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). We will keep the lists of all active nodes, A, and all orphans, O. The general structure of the algorithm is:

```
initialize:  S = {s}, T = {t}, A = {s,t}, O = ∅
while true
        grow S or T 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. It is convenient to store content of search trees S and T via flags $TREE(p)$ indicating affiliation of each node p so that

$$TREE(p) = \begin{cases} S & if\ p \in S \\ T & if\ p \in T \\ \emptyset & if\ p\ is free \end{cases}$$

If node p belongs to one of the search trees then the information about its parent will be stored as $PARENT(p)$. Roots of the search trees (the source and the sink), orphans, and all free nodes have no parents, t.e. $PARENT(p) = \emptyset$. We will also use notation $tree\_cap(p \to q)$ to describe residual capacity of either edge (p, q) if $TREE(p) = S$ or edge (q, p) if $TREE(p) = T$. These edges should be non-saturated in order for node p to be a valid parent of its child q depending on the search tree.

3.2.1 Growth stage:

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

```
while A ≠ ∅
    pick an active node p ∈ A
    for every neighbor q such that tree_cap(p → q) > 0
        if TREE(q) = ∅ then add q to search tree as an active node:
            TREE(q) := TREE(p),  PARENT(q) := p,  A := A ∪ {q}
        if TREE(q) ≠ ∅ and TREE(q) ≠ TREE(p) return P = PATH_{s→t}
    end for
    remove p from A
end while
return P = ∅
```

3.2.2 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 might 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
    if TREE(p) = TREE(q) = S then set PARENT(q) := ∅ and O := O ∪ {q}
    if TREE(p) = TREE(q) = T then set PARENT(p) := ∅ and O := O ∪ {p}
end for
```

3.2.3 Adoption stage:

During this stage all orphan nodes in O are processed until O becomes empty. Each node p being processed tries to find a new valid parent within the same search tree; in case of success p remains in the tree but with a new parent, otherwise it becomes a free node and all its children are added to O.

```
while O ≠ ∅
    pick an orphan node p ∈ O and remove it from O
    process p
end while
```

The operation "`process` p" consists of the following steps. First we are trying to find a new valid parent for p among its neighbors. A valid parent q should satisfy: $TREE(q) = TREE(p)$, $tree\_cap(q \to p) > 0$, and the "origin" of q should be either source or sink. Note that the last condition is necessary because during adoption stage some of the nodes in the search trees S or T may originate from orphans.

If node p finds a new valid parent q then we set $PARENT(p) = q$. In this case p remains in its search tree and the active (or passive) status of p remains unchanged. If p does not find a valid parent then p becomes a free node and the following operations are performed:

- scan all neighbors q of p such that $TREE(q) = TREE(p)$:
  - if $tree\_cap(q \rightarrow p) > 0$ add q to the active set A
  - if $PARENT(q) = p$ add q to the set of orphans O and set $PARENT(q) := \emptyset$
- $TREE(p) := \emptyset,\ \ A := A - \{p\}$

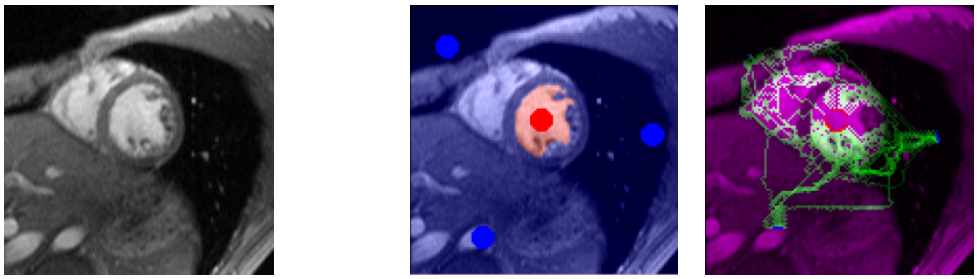
Note that as p becomes free all its neighbors connected through non-saturated edges should became active. It may happen that some neighbor q did not qualify as a valid parent during adoption stage because it did not originate from the source or the sink. However, this node could be a valid parent after adoption stage is finished. At this point q must have active status as it is located next to a free node p.

3.3 Algorithm tuning

The proof of correctness of the algorithm presented above is straightforward (see [19]). At the same time, our description leaves many free choices in implementing certain details. For example, we found that the order of processing active nodes and orphans may have a significant effect on the algorithm's running time. Our preferred processing method is a very minor variation of "First-In-First-Out". In this case the growth stage can be described as a breadth-first search. This guarantees that at least the first path from the source to the sink is the shortest. Note that the search tree may change unpredictably during adoption stage. Thus, we can not guarantee anything about paths found after the first one.

There are many additional free choices in implementing the adoption stage. For example, as an orphan looks for a new parent it has to make sure that a given candidate is connected to the source or to the sink. We found that "marking" nodes confirmed to be connected to the source at a given adoption stage helps to speed up the algorithm. In this case other orphans do not have to trace the roots of their potential parents all the way to the terminals. We also found that keeping distance-to-source information in addition to these "marks" allows orphans to select new parents that are closer to the source. This further helps with the algorithm's speed because we get shorter paths.

We used a fixed tuning of our algorithm in all experiments of Section 4 and a library with our implementation is available upon request for research purposes. The general goal of tuning



Original image (a) Graph cut (b) Graph flow

Figure 4: Graph cut/flow example in the context of image segmentation in Section 4.4. The cost of edges between the pixels is low in places with high intensity contrast. Red and blue seeds are linked to the source s and the sink t, correspondingly. (a) shows the optimal segmentation (minimum cut) and (b) displays augmenting paths (flow) from s to t. As the flow increases it quickly saturates weak edges that are at the boundary of segments in (a).

was to make augmenting paths as short as possible. Note that augmenting paths on graphs in vision can be easily visualized. In the majority of cases such graphs are regular grids of nodes that correspond to image pixels. Then, augmenting paths and the whole graph flow can be meaningfully displayed (see Figure 4). We can also display the search tree at different stages. This allows a very intuitive way of tuning max-flow methods in vision.

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 4.2), stereo (Section 4.3), and object segmentation (Section 4.4). We chose formulations where certain appropriate versions of energy (1) can be minimized via graph cuts. The corresponding graph structures were previously described by [13, 15, 3, 6, 20, 21, 4] in detail. These (or very similar) structures are used in all computer vision applications with graph cuts (that we are aware of) to date.

4.1 Experimental Setup

Note that we could not test all known min-cut/max-flow algorithms. In our experimental tests on graph-based energy minimization methods in vision we compared the new algorithm in Section 3

and the following standard min-cut/max-flow algorithms outlined in Section 2.2:

DINIC: Algorithm of Dinic [10].

H_PRF: Push-Relabel algorithm [12] with the highest level selection rule.

Q_PRF: Push-Relabel algorithm [12] with the queue based selection rule.

Many previous experimental tests, including the results in [8], 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.

For DINIC, H_PRF, and Q_PRF we used the implementations written by Cherkassky and Goldberg [8], except that we converted them from C to C++ style and modified the interface (i.e. functions for creating a graph). Both H_PRF and Q_PRF use global and gap relabeling heuristics. Our algorithm was implemented in C++. We selected a tuning described in Section 3.3 with more details available in [19]. We did not make any machine specific optimization (such as pipeline-friendly instruction scheduling or cache-friendly memory usage).

Experiments in sections 4.2 and 4.4 were performed on 1.4GHz Pentium IV PC (2GB RAM, 8KB L1 cache, 256KB L2 cache), and experiments in Section 4.3 were performed on UltraSPARC II workstation with four 450 MHz processors and 4GB RAM. In the former case we used Microsoft Visual C++ 6.0 compiler, Windows NT platform and in the latter case - GNU C++ compiler, version 3.2.2 with the flag "-O5", SunOS 5.8 platform. To get system time, we used ftime() function in Unix and _ftime() function in Windows. Although these functions do not measure process computation time, we felt that they were appropriate since we got very consistent results (within 1%) when running tests multiple times.

4.2 Image Restoration

Image restoration is a representative early vision problem. The goal is to restore original pixel intensities from the observed noisy data. Some examples of image restoration are shown in Figure 5. The problem can be very easily formulated in terms of energy (1) minimization. In

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

Figure 5: Image Restoration Examples

fact, many other low level vision problems can be represented by the same energies. We chose the context of image restoration mainly for its simplicity.

In this section we consider two examples of energy (1) based on the Potts and linear models of interaction, correspondingly. Besides image restoration [13], graph methods for minimizing Potts energy were used in segmentation [18], stereo [3, 6], object recognition [2], shape reconstruction [24], and augmented reality [26]. Linear interaction energies were used in stereo [22] and segmentation [15]. Minimization of the linear interaction energy is based on graphs that are quite different from what is used for the Potts model. At the same time, there is very little variation between the graphs in different applications when the same type of energy is used. They mainly differ in their specific edge cost settings while the topological properties of graphs are almost identical once the energy model is fixed.

4.2.1 Potts Model

The Potts energy that we use for image restoration is

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

where $I = \{I_p \,|p \in \mathcal{P}\}$ is a vector of unknown "true" intensities of pixels in image $\mathcal{P}$ and $I^o = \{I_p^o \,|p \in \mathcal{P}\}$ are observed intensities corrupted by noise. The Potts interactions are specified by penalties $K_{(p,q)}$ for intensity discontinuities between 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. [13].

We consider image restoration with multiple labels where the problem becomes NP hard. We use iterative α-*expansion* method in [6] which is guaranteed to find a solution within a factor of two from the global minimum of the Potts energy. At a given iteration [6] allows any subset of pixels to switch to a fixed label α. In fact, the algorithm finds an optimal subset of pixels that gives the largest decrease in the energy. The computation is done via graph cuts using some generalization of the basic graph structure in [13] (see Figure 2). The algorithm repeatedly cycles through all possible labels α until no further improvement is possible.

Two tables below present the running times (in seconds, 1.4GHz Pentium IV) when different max-flow/min-cut algorithms are employed in the basic step of each α-expansion. Each table corresponds to one of the original images shown in Figure 5. The number of allowed labels is 210 (*Diamond*) and 244 (*Bell Quad*), correspondingly. We run the algorithms on images at different resolutions. At each column we state the exact size (HxW) in pixels. Note that the total number of pixels increases by a factor of 2 from left to right.

method	**input:** Diamond, 210 labels						
	35x35	50x50	70x70	100x100	141x141	200x200	282x282
DINIC	0.39	0.77	3.42	4.19	13.85	43.00	136.76
H_PRF	0.17	0.34	1.16	1.68	4.69	12.97	32.74
Q_PRF	0.16	0.35	1.24	1.70	5.14	14.09	40.83
Our	0.16	0.20	0.71	0.74	2.21	4.49	12.14

method	**input:** Bell Quad, 244 labels					
	44x44	62x62	87x87	125x125	176x176	250x250
DINIC	1.32	4.97	13.49	37.81	101.39	259.19
H_PRF	0.31	0.72	1.72	3.85	8.24	18.69
Q_PRF	0.20	1.00	1.70	4.31	10.65	25.04
Our	0.19	0.48	0.98	2.11	4.84	10.47

Note that the running times above correspond to the end of the first cycle of the α-expansion method in [6] when all labels were expanded once. The relative speeds of different max-flow/min-cut algorithms do not change much when the energy minimization is run to convergence. The number of cycles it takes to converge can vary from 1 to 3 for different resolutions/images. Thus, the running times to convergence are hard to compare between the columns and we do not present them. In fact, restoration results are quite good even after the first iteration. In most cases additional iterations do not improve the actual output much. Figure 5(a) shows the result of the Potts model restoration of the *Diamond* image (100x100) after the first cycle of iterations.

4.2.2 Linear interaction energy

Here we consider image restoration with "linear" interaction energy. Figure 5(c) shows one restoration result that we obtained in our experiments with this energy. The linear interaction energy can be written as

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

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 [15, 3]. In fact, both of these methods use graphs that are very similar to the one introduced by [22] in the context of multi-camera stereo. The graphs are constructed by consecutively connecting multiple layers of image-grids. Each layer corresponds to one label. The two terminals are connected only to the first and the last layers. Note that the topological structure of these graphs is noticeably different from the Potts model graphs especially when the number of labels (layers) is large.

The tables below show the running times (in seconds on 1.4GHz, Pentium IV) that different min-cut/max-flow algorithms took to compute the exact minimum of the linear interactions energy (3). We used the same *Diamond* and *Bell Quad* images as in the Potts energy tests. We run the algorithms on images at different resolution. At each column we state the exact size (hight and width) in pixels. Note that the total number of pixels increases by a factor of 2 from left to right.

method	**input:** Diamond, 54 labels					
	35x35	50x50	70x70	100x100	141x141	200x200
DINIC	1.34	4.13	8.86	18.25	34.84	57.09
H_PRF	0.47	1.30	3.03	7.48	17.53	43.58
Q_PRF	0.55	1.16	3.05	6.50	12.77	22.48
Our	0.17	0.33	0.63	1.41	2.88	5.98

method	**input:** Bell Quad, 32 labels					
	44x44	62x62	87x87	125x125	176x176	250x250
DINIC	0.55	1.25	2.77	6.89	15.69	31.91
H_PRF	0.48	1.25	2.75	7.42	17.69	38.81
Q_PRF	0.27	0.56	1.55	2.39	6.78	10.36
Our	0.13	0.27	0.52	1.09	2.33	4.84

The structure of linear interaction graph directly depends on the number of labels[4]. In fact, if there are only two labels then the graph is identical to the Potts model graph. However, both size and topological properties of the linear interaction graphs change as the number of labels (layers) gets larger and larger. Below we compare the running times of the algorithms for various numbers of allowed labels (layers). We consider the same two images, *Diamond* and *Bell Quad*. In each case the size of the corresponding image is fixed. At each column we state the number of allowed labels $\mathcal{L}$. The number of labels increases by a factor of 2 from left to right.

method	**input:** Diamond, 100x100 (pix)				**input:** Bell Quad, 125x125 (pix)			
	$\mathcal{L}$=27	$\mathcal{L}$=54	$\mathcal{L}$=108	$\mathcal{L}$=215	$\mathcal{L}$=32	$\mathcal{L}$=63	$\mathcal{L}$=125	$\mathcal{L}$=250
DINIC	6.89	18.16	50.81	166.27	6.91	17.69	46.64	102.74
H_PRF	3.05	7.38	15.50	47.49	7.47	19.30	58.14	192.39
Q_PRF	2.36	6.41	17.22	43.47	2.39	7.95	15.83	45.64
Our	0.55	1.39	4.34	16.81	1.13	2.95	10.44	41.11

Our experiments with linear interaction graphs show that most of the tested max-flow/min-cut algorithms are close to linear with respect to increase in image size. At the same time, none of the algorithms behaved linearly with respect to the number of labels despite the fact that the size of graphs linearly depend on the number of labels. Our algorithm is a winner in absolute speed as in most of the tests it is 2-4 times faster than the second best method. However, our algorithm's dynamics with respect to increase in the number of labels is not favorable. For example, Q_PRF gets real close to the speed of our method in case of $\mathcal{L}$=250 (*Bell Quad*) even though our algorithm was 2 times faster than Q_PRF when the number of labels was $\mathcal{L}$=32.

4.3 Stereo

Stereo is another classical vision problem where graph-based energy minimization methods have been successfully applied. The goal of stereo is to compute the correspondence between pixels of two or more images of the same scene obtained by cameras with slightly different view points. We consider three graph-based methods for solving this problem: pixel-labeling stereo with the Potts model [3, 6], stereo with occlusions [20], and multi-camera scene reconstruction [21]. Note that the last method is designed for a generalization of the stereo problem to the case of more than two cameras.

[4]Note that in Section 4.2.1 we tested multi-label Potts energy minimization algorithm [6] where the number of labels affects the number of iterations but has no effect on the graph structures.

4.3.1 Pixel-labeling stereo with the Potts model

First, we consider a formulation of stereo problem given in [3, 6] which is practically identical to our formulation of the restoration problem in Section 4.2.1. We seek a disparity labeling $d = \{d_p | p \in \mathcal{P}\}$ which minimizes the energy

$$E(d) = \sum_{p \in \mathcal{P}} D(p, d_p) + \sum_{(p,q) \in \mathcal{N}} K_{(p,q)} \cdot T(d_p \neq d_q) \tag{4}$$

where d_p is a disparity label of pixel p in the left image, and $D(p, d)$ is a penalty for assigning a label d to a pixel p (the squared difference in intensities between corresponding pixels in the left and in the right images). We use the same iterative α-expansion method from [6] as in the restoration section above.

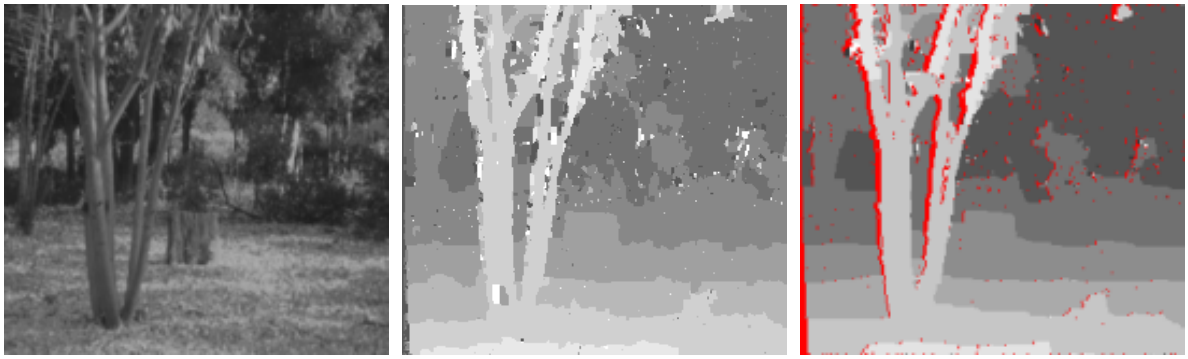
The tests were done on three stereo examples shown in Figure 6. We used the *Head* pair from the University of Tsukuba, and the well-known *Tree* pair from SRI. To diversify our tests we compared the speed of algorithms on a *Random* pair where the left and the right images did not correspond to the same scene (they were taken from the *Head* and the *Tree* pairs, respectively).

Running times for the stereo examples in Figure 6 are shown in seconds (450 MHz Ultra-SPARC II Processor) in the table below. As in the restoration section, the running times correspond to the first cycle of the algorithm. The relative performance of different max-flow/min-cut algorithms is very similar when the energy minimization is run to convergence while the number of cycles it takes to converge varies between 3 and 5 for different data sets. We performed two sets of experiments: one with a four-neighborhood system and the other with an eight-neighborhood system. The corresponding running times are marked by "N4" and "N8". The disparity maps at convergence are shown in Figure 6(b,e,h). The convergence results are slightly better than the results after the first cycle of iterations. We obtained very similar disparity maps in N4 and N8 cases.

method	*Head*, 384x288 (pix)		*Tree*, 256x233 (pix)		*Random*, 384x288 (pix)	
	N4	N8	N4	N8	N4	N8
DINIC	104.18	151.32	9.53	19.80	105.93	167.16
H_PRF	12.00	18.03	1.65	2.86	14.25	18.22
Q_PRF	10.40	14.69	2.13	3.33	12.05	15.64
Our	3.41	6.47	0.68	1.42	3.50	6.87

(a) Left image of *Head* pair (b) Potts model stereo (c) Stereo with occlusions

Disparity maps obtained for the *Head* pair



(d) Left image of *Tree* pair (e) Potts model stereo (f) Stereo with occlusions

Disparity maps obtained for the *Tree* pair

(g) *Random* pair (h) Potts model stereo (i) Stereo with occlusions

Disparity maps obtained for the *Random* pair

Figure 6: Stereo results. The sizes of images are 384×288 in (a-c), 256×233 in (d-f), and 384×288 in (g-i). The results in (c,f,i) show occluded pixels in red color.

4.3.2 Stereo with Occlusions

Any stereo images of multi-depth objects contain occluded pixels. The presence of occlusions adds significant technical difficulties to the problem of stereo as the space of solutions needs to be constrained in a very intricate fashion. Most stereo techniques ignore the issue to make the problem tractable. Inevitably, such simplification can generate errors that range from minor inconsistencies to major misinterpretation of the scene geometry. Recently, [1] reported some progress in solving stereo with occlusions. [14] were first to suggest a graph-cut based solution for stereo that elegantly handles occlusions assuming monotonicity constraint.

Here we consider a more recent graph-based formulation of stereo [20] that takes occlusions into consideration without making extra assumptions about scene geometry. 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). [20] 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 [20] represent *pairs* rather than pixels or voxels.

We used the same three datasets as in the previous section. Running times for these stereo examples in Figure 6 are shown in seconds (450 MHz UltraSPARC II Processor) in the table below. The times are for the first cycle of the algorithm. Algorithm results after convergence are shown in Figure 6(c,f,i).

method	*Head*, 384x288 (pix)		*Tree*, 256x233 (pix)		*Random*, 384x288 (pix)	
	N4	N8	N4	N8	N4	N8
DINIC	376.70	370.94	66.19	102.60	81.70	115.58
H_PRF	35.65	49.81	9.07	15.41	5.48	8.32
Q_PRF	33.12	44.86	8.55	13.64	9.36	14.02
Our	10.64	19.14	2.73	5.51	3.61	6.42

4.3.3 Multi-camera scene reconstruction

In this section we consider a graph cuts based algorithm for reconstructing a shape of an object taken by several cameras [21].

Suppose we are given n calibrated images of the same scene taken from different viewpoints (or at different moments of time). Let $\mathcal{P}_i$ be the set of pixels in the camera i, and let $\mathcal{P} = \mathcal{P}_1 \cup \ldots \cup \mathcal{P}_n$ be the set of all pixels. A pixel $p \in \mathcal{P}$ corresponds to a ray in 3D-space. Consider the point of the first intersection of this ray with an object in the scene. Our goal is to find the depth of this point for all pixels in all images. Thus, we want to find a labeling $f : \mathcal{P} \to \mathcal{L}$ where $\mathcal{L}$ is a discrete set of labels corresponding to different depths. We tested the algorithm for image sequences with labels corresponding to parallel planes in 3D-space.

A pair $\langle p, l \rangle$ where $p \in \mathcal{P}$, $l \in \mathcal{L}$ corresponds to some point in 3D-space. We will refer to such pairs as *3D-points*. The set of interactions I will consist of (unordered) pairs of 3D-points with the same label $\langle p_1, l \rangle$, $\langle p_2, l \rangle$ "close" to each other in 3D-space.

We minimize the energy function consisting of three terms:

$$E(f) = E_{data}(f) + E_{smoothness}(f) + E_{visibility}(f) \tag{5}$$

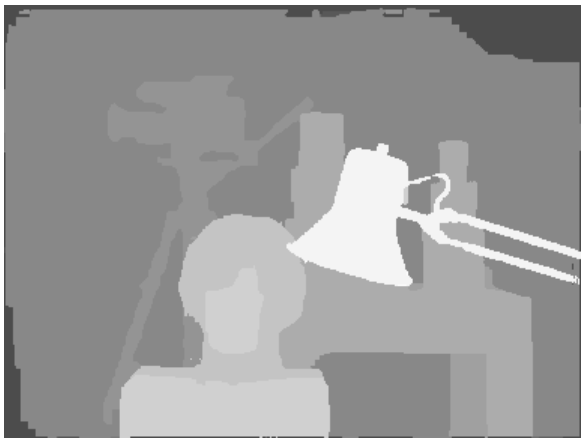
The data term imposes photo-consistency. It is

$$E_{data}(f) = \sum_{\langle p, f(p) \rangle, \langle q, f(q) \rangle \in I} D(p, q)$$

where $D(p, q)$ is a non-positive value depending on intensities of pixels p and q (for example, $D(p, q) = min\{0, (Intensity(p) - Intensity(q))^2 - K\}$ for some constant $K > 0$).

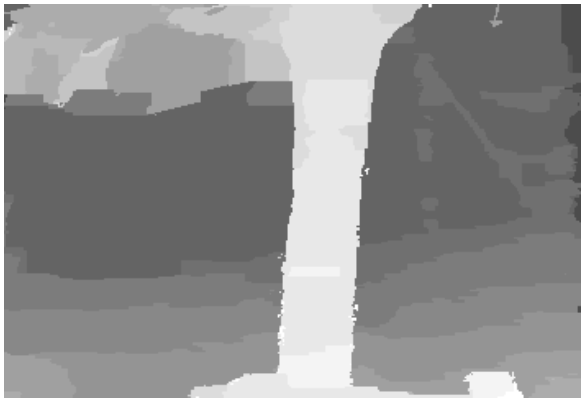
The smoothness term is the sum of Potts energy terms over all cameras. The visibility term is infinity if a configuration f violates the visibility constraint, and zero otherwise. More details can be found in [21].

(a) Middle image of *Head* dataset



(b) Scene reconstruction for *Head* dataset

(c) Middle image of *Garden* sequence



(d) Scene reconstruction for *Garden* sequence

(e) Middle image of *Dayton* sequence

(f) Scene reconstruction for *Dayton* sequence

Figure 7: Multi-camera reconstruction results. There are 5 images of size 384×288 in (a), 8 images of size 352×240 in (c) and 5 images of size 384×256 in (e).

The tests were done for three datasets: the *Head* sequence from the University of Tsukuba, the *Garden* sequence and the *Dayton* sequence. Middle images of these datasets are shown in Figure 7. The table below gives running times (in seconds, 450 MHz UltraSPARC II Processor) for these three datasets. The times are for the first cycle of the algorithm. Algorithm results after three cycles are shown in Figure 7(b,d,f).

method	**input sequence**		
	Head, 5 views 384x288	*Garden*, 8 views 352x240	*Dayton*, 5 views 384x256
DINIC	2793.48	2894.85	2680.91
H_PRF	282.35	308.52	349.60
Q_PRF	292.93	296.48	266.08
Our	104.33	81.30	85.56

4.4 Segmentation

In this section we describe experimental tests that compare running times of the selected min-cut/max-flow algorithms for segmentation technique in [4]. The graph-based method in [4] allows to segment an object of interest in N-D images/volumes. This technique generalizes the MAP-MRF method of Greig at. al. [13] by incorporating additional hard constraints into 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. The hard constraints may come from a user placing object and background seeds.

The technique finds a globally optimal binary segmentation of N-dimensional image satisfying the hard constraints (seeds). The computation is done in one pass of a max-flow/min-cut algorithm on a certain graph. In case of 2D images the structure of the underlying graph is exactly the same as shown in Figure 2. In 3D cases [4] build a regular 3D grid graph.

We tested min-cut/max-flow algorithms on 2D and 3D segmentation examples illustrated in Figure 8. This figure demonstrates original data and our segmentation results corresponding to some sets of seeds. Note that the user can place seeds interactively. New seeds can be added to correct segmentation imperfections. The technique in [4] efficiently recomputes the optimal solution starting at the previous segmentation result.

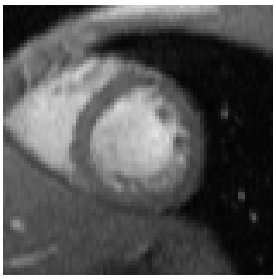
Figure 8(a-b) shows one of our experiments where a group of people around a bell were

Photo Editing

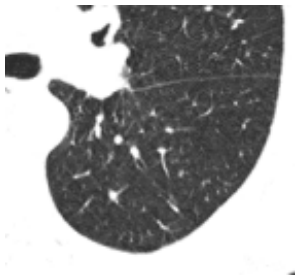
(a) Bell Photo

(b) Bell Segmentation

Medical Data

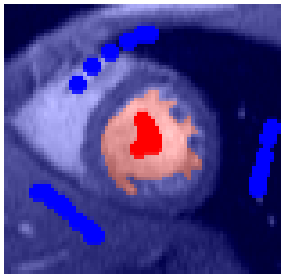


(c) Cardiac MR

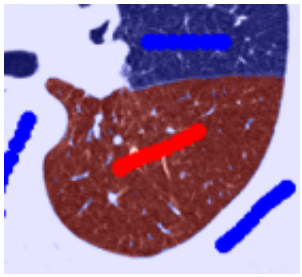


(e) Lung CT

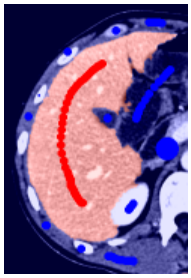
(g) Liver MR



(d) LV Segment



(f) Lobe Segment



(h) Liver Segment

Figure 8: Segmentation Experiments

segmented on a real photo image (255x313 pixels). Other segmentation examples in (c-h) are for 2D and 3D medical data. In (c-d) we segmented a left ventricle in a 3D cardiac MR data (127x127x12 voxels). In our 3D experiments the seeds were placed in one slice in the middle of the volume. Often, this is enough to segment the whole volume correctly. The tests with lung CT data (e-f) were made in 2D (409x314 pixels) case. The goal was to segment out a lower lung lobe. In (g-h) we tested the algorithms on 2D liver MR data (511x511 pixels). Additional 3D experiments were performed on heart ultrasound and kidney MR volumes.

The tables below compare running times (in seconds, 1.4GHz Pentium IV) of the selected min-cut/max-flow algorithms for a number of segmentation examples. Note that these times include only min-cut/max-flow computation[5]. In each column we show running times of max-flow/min-cut algorithms corresponding to exactly the same set of seeds. The running times were obtained for the "6" and "26" neighborhood systems (N6 and N26). Switching from N6 to N26 increases complexity of graphs but does not affect the quality of segmentation results much.

method	**2D examples**					
	Bell photo (255x313)		Lung CT (409x314)		Liver MR (511x511)	
	N4	N8	N4	N8	N4	N8
DINIC	2.73	3.99	2.91	3.45	6.33	22.86
H_PRF	1.27	1.86	1.00	1.22	1.94	2.59
Q_PRF	1.34	0.83	1.17	0.77	1.72	3.45
Our	0.09	0.17	0.22	0.33	0.20	0.45

method	**3D examples**					
	Heart MR (127x127x12)		Heart US (76x339x38)		Kidney MR (99x66x31)	
	N6	N26	N6	N26	N6	N26
DINIC	20.16	39.13	172.41	443.88	3.39	8.20
H_PRF	1.38	2.44	18.19	47.99	0.19	0.50
Q_PRF	1.30	3.52	23.03	45.08	0.19	0.53
Our	0.70	2.44	13.22	90.64	0.20	0.58

5 Conclusions

We tested a reasonable sample of typical vision graphs. In most examples our new min-cut/max-flow algorithm worked 2-5 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

[5]Time for entering seeds may vary between different users. For the experiments in Figure 8 all seeds were placed within 10 to 20 seconds.

some cases the new algorithm made possible near real-time performance of the corresponding applications.

More specifically, we can conclude that our algorithm is consistently several times faster (than the second best method) in all applications where graphs are 2D grids. However, our algorithm is not a clear out-performer when complexity of underlying graphs is increased. For example, linear interaction energy graphs (Section 4.2.2) with a large number of grid-layers (labels) is one example where Q_PRF performance was comparable to our algorithm. Similarly, experiments in Section 4.4 show that push-relabel methods (H_PRF and Q_PRF) are comparable to our algorithm in 3D segmentation tests even though it was several times faster in all 2D segmentation examples. Going from the "6" neighborhood system to the "26" system further decreased relative performance of our method in 3D segmentation.

Note that we do not have a polynomial bound for our algorithm[6]. Interestingly, in all our practical tests on 2D and 3D graphs that occur in real computer vision applications our algorithm significantly outperformed a polynomial method of DINIC. Our results suggest that grid graphs in vision are a very specific application for min-cut/max-flow algorithms. In fact, Q_PRF outperformed H_PRF in many of our tests (especially in Section 4.2.2) despite the fact that H_PRF is generally regarded as the fastest algorithm in combinatorial optimization community.

Acknowledgments

Most of our work was done at Siemens Research, NJ, and it would not be possible without general support from Alok Gupta and Gareth Funka-Lea. We would like to thank Olga Veksler (NEC Research, NJ) who greatly helped with implementations for Section 4.2. We also thank Ramin Zabih (Cornell, NY) for the discussions that helped to improve our paper.

[6]The trivial bound given in Section 3 involves the cost of a minimum cut and, theoretically, it is not a polynomial bound. In fact, additional experiments showed that our algorithm is by several orders of magnitude slower than Q_PRF, H_PRF, and DINIC on several standard (outside computer vision) types of graphs commonly used for tests in combinatorial optimization community.

References

[1] A. F. Bobick and S. S. Intille. Large occlusion stereo. *International Journal of Computer Vision*, 33(3):181–200, September 1999.

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

[3] 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.

[4] 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*, volume I, pages 105–112, July 2001.

[5] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition (EMMCVPR)*, number 2134 in LNCS, pages 359–374, Sophia Antipolis, France, September 2001. Springer-Verlag.

[6] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, November 2001.

[7] Chris Buehler, Steven J. Gortler, Michael F. Cohen, and Leonard McMillan. Minimal surfaces for stereo. In *7th European Conference on Computer Vision*, volume III of *LNCS 2352*, pages 885–899, Copenhagen, Denmark, May 2002. Springer-Verlag.

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

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

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

[11] L. Ford and D. Fulkerson. *Flows in Networks.* Princeton University Press, 1962.

[12] 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.

[13] 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.

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

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

[16] Hiroshi Ishikawa. Exact optimization for markov random fields with convex priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2003, to appear.

[17] David R. Karger. Random sampling in cut, flow, and network design problems. *Mathematics of Operations Research*, 24(2):383–413, May 1999.

[18] 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 *f*MRI data analysis. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 62–71, 2000.

[19] Vladimir Kolmogorov. *Graph-based Algorithms for Multi-camera Reconstruction Problem.* PhD thesis, Cornell University, CS Department, 2003, to appear.

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

[21] Vladimir Kolmogorov and Ramin Zabih. Multi-camera scene reconstruction via graph cuts. In *7th European Conference on Computer Vision*, volume III of *LNCS 2352*, pages 82–96, Copenhagen, Denmark, May 2002. Springer-Verlag.

[22] 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.

[23] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 2002, to appear. An earlier version appears in CVPR 2001 Workshop on Stereo Vision.

[24] 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.

[25] Richard Szeliski and Ramin Zabih. An experimental comparison of stereo algorithms. In *Vision Algorithms: Theory and Practice*, number 1883 in LNCS, pages 1–19, Corfu, Greece, September 1999. Springer-Verlag.

[26] 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.

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