Graph Cuts in Vision and Graphics: Theories and Applications

Yuri Boykov and Olga Veksler

Computer Science, The University of Western Ontario, London, Canada
[yuri,olga]@csd.uwo.ca

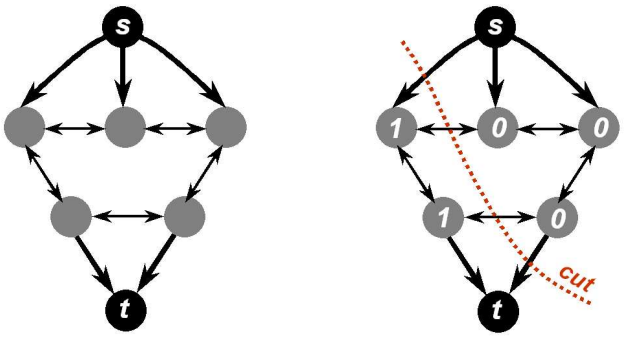
Abstract. Combinatorial min-cut algorithms on graphs emerged as an increasingly useful tool for problems in vision. Typically, the use of graph-cuts is motivated by one of the following two reasons. Firstly, graph-cuts allow geometric interpretation; under certain conditions a cut on a graph can be seen as a hypersurface in N-D space embedding the corresponding graph. Thus, many applications in vision and graphics use min-cut algorithms as a tool for computing optimal hypersurfaces. Secondly, graph-cuts also work as a powerful energy minimization tool for a fairly wide class of binary and non-binary energies that frequently occur in early vision. In some cases graph cuts produce globally optimal solutions. More generally, there are iterative graph-cut based techniques that produce provably good approximations which (were empirically shown to) correspond to high-quality solutions in practice. Thus, another large group of applications use graph-cuts as an optimization technique for low-level vision problems based on global energy formulations.

This chapter is intended as a tutorial illustrating these two aspects of graph-cuts in the context of problems in computer vision and graphics. We explain general theoretical properties that motivate the use of graph cuts, as well as, show their limitations. Our goal is to familiarize the reader with graph-cuts as a generic tool for optimization in vision.

1 Graph Cuts Basics

In this section we introduce the basic terminology and review the minimum cut max flow problem. Let $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ be a graph which consists of a set of nodes $\mathcal{V}$ and a set of directed edges $\mathcal{E}$ that connect them. The nodes set $\mathcal{V}$ contains two special *terminal* nodes which are called the *source*, s, and the *sink*, t. In Figure 1(a) we show a simple example of a graph with the terminals s and t.

All edges in the graph are assigned some nonnegative weight or cost $w(p, q)$. A cost of a directed edge (p, q) may differ from the cost of the reverse edge (q, p). We can break all edges in the graph in two groups: n-links and t-links. An edge is called a t-link if it connects a non-terminal node with a terminal node. An edge is called an n-link if it connects a pair of nonterminal nodes. T-links are shown with thicker lines and n-links are shown with thiner lines in figure 1(a).



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

Fig. 1. Example of a graph with terminal nodes s and t. To depict a pair of edges (p, q) and (q, p), we use one link with arrows at both ends.

1.1 The Min-Cut and Max-Flow Problem

An s/t cut C (sometimes we just call it a *cut*) 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 $q \in \mathcal{T}$. If (p, q) is such a boundary edge, then we sometimes say that cut C severs edge (p, q).The *minimum cut* problem on a graph is to find a cut that has the minimum cost among all possible 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. Speaking informally, 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. Thus, min-cut and max-flow problems are equivalent. In fact, the maximum flow value is equal to the cost of the minimum cut.

1.2 Algorithms for the Min-Cut and Max-Flow Problem

There are many standard polynomial time algorithms for min-cut/max-flow[10]. These algorithms can be divided into two main groups: "push-relabel" style methods [13] and algorithms based on Ford-Fulkerson style augmenting paths. In practice the push-relabel style algorithms perform better for general graphs. In vision application, however, the most common type of a graph is a two or a higher dimensional grid. For the grid graphs, Boykov and Kolmogorov [7] developed a fast augmenting path based algorithm which significantly outperforms the push

relabel algorithm. Furthermover, the observed running time of the algorithm in [7] is linear.

While the algorithm in [7] is very efficinet, with the execution time of only a few seconds for a typical problem, it is still far from real time. In recent years, GPU accelaration has become popular for improving the efficiency. Not every algorithm is suitable for a GPU implementation. Among the existing graph cut algorithms, it is feasible to implement the push relabel algorithm on a grid graph for a GPU. This is a promising direction to move the graph cuts closer to a real time implementation.

2 Graph Cuts as Binary Optimization

In this section we show that the minimum graph cut algorithm is a powerful optimization tool which is inherently binary. It is particularly suitable for vision applications because it can be used for enforcing piecewise coherance. To illustrate the concepts in this section, we provide the earliest example of the graph cuts use in vision, which also happens to be particularly clear.

Consider Figure 1(b) again. Recall that a graph cut is a partition of all nonterminal nodes into two sets, $\mathcal{S}$ and $\mathcal{T}$. Let each non termnal node $p \in \mathcal{V}$ represent a binary variable, and let us associate 0 with the source s and 1 with the sink t. Then we can identify any cut with a particular assignment to binary variables (i.e. nodes) in the obvious way: if node $p \in \mathcal{S}$ then its binary value is 0, and if node $p \in \mathcal{T}$ then its binary value is 1. This procedure establishes a one to one correspondence between the set of all variable assignments and the set of graph cuts. In figure 1(b) we label each node with the binary value corresponding to the given cut. If n is the number of nonterminal nodes in the graph, then there are 2^n graph cuts. The cost of each cut is

$$c(\mathcal{S}, \mathcal{T}) = \sum_{p \in \mathcal{S}, q \in \mathcal{T}} w(p, q), \tag{1}$$

and it can be used as the objective function to evaluate the quality of partition $C = \{\mathcal{S}, \mathcal{T}\}$ or, equivalently, as the energy of the corresponding variable assignment to the nonterminal nodes. Then the minimum cut corresponds to the optimal variable assignment for the nodes. Thus the power of the graph cuts algorithm is that it can search efficiently through the exponential space of solutions and find an optimum one in polynomial time. In this sense graph cuts are similar to dynamic programming which also can find the optimum in an exponential space of solutions in polynomial time. The significant disadvantage of dynamic programming is that it can only be applied to tree graphs, while graph cuts can be applied to arbitrary graphs. The biggest advantage of dynamic programming is that it can be used with arbitrary objective functions, whereas the objective functions that graph cuts can minimize are limited.

What makes graph cuts especially attractive for vision applications is their regularization properties. In vision, most problems are ill defined, if only the

constraints derived from data are used. Thus additional or "regularizing" constraints are needed to find an acceptable solution. One natural way of regularizing a vision problem is to require the solution to be piecewise smooth or piecewise constant. Consider a graph where each node is connected by an edge to a few nearby nodes, which are typically called *neighbors*. By minimizing the sum of edge weights that go between $\mathcal{S}$ and $\mathcal{T}$, graph cuts algorthm is looking for a piecewise constant partitioning, that is a partitioning where most neighbors of a node p are assigned the same binary value as the one assigned to p. An equivalent way of describing the regularizing properties of graph cuts is that it is seeks the solution with the smallest boundary cost. Typically, the regularization constraints are encoded in the n-link weights, and the data constraints are encorporated into the t-link weights. Thus if we split the sum over all edges in equation 1 into two sums, the first sum over the t-links, and in the second sum over the n-links, then the objective function becomes a typical one used in regularization. The first term of the objective function comes from the data, and the second term is the regularization term.

From the discussion above it should be clear that graph cuts can be used to optimize those binary energy functions that, intuitively, can be represented by a graph. We can start by constructing a graph such that the corresponding binary energy function is appropriate for our problem. Alternatively, we can start by designing an appropriate binary energy function and then ask whether it can be minimized by graph cuts. When using the second approach, we need the answer to the question of what kind of binary energy functions can in fact be optimized with graph cuts? Recently, Kolmogorov and Zabih in their major contribution to the field [22], have given the answer to this question by characterizing precisely the class of binary energy functions that can be minimized with graph cuts. What makes their work even more significant is that their proof is constructive. Thus for a given binary energy function, one has to check the so called "regularity" condition. If it does not hold, then the binary energy function in question cannot be minimized with graph cuts. If the regularity condition holds, then the paper gives a simple algorithm to construct the corresponding graph. Thus if we wish to solve a problem by binary energy minimization, the results in [22] elliminate the guesswork in designing an appropriate graph. There are many interesting problems which can be reduced to regular binary energy functions, as we shall see in secton ???.

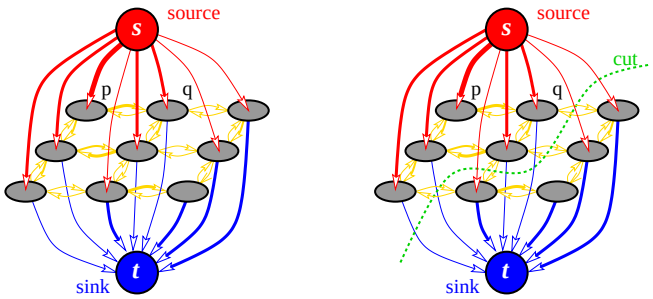
It is important not to make a mistake by viewing the results in [22] as negative. One may be tempted to say that graph cuts are very limited in application since they can only minimize regular binary energy functions. However graph cuts can be used to find an approximate solution with strong optimality guarantees for a large class of non binary energy functions, as we shall see in section 4. To optimize a nonbinary energy function, an iterative scheme is used. At each iteration, a binary energy function is minimized, and at that step, the results in [22] are of great help.

It is interesting to note that the results in [22] are only about binary energy functions, that is each nonterminal node represents a binary variable. However,

the nodes of a graph can be used to encode an energy function which is not binary, by using several nodes to represent a single nonbinary variable. In this case, the results in [22] do not apply. What kind of nonbinary energy functions can be optimized exactly with graph cuts is still an open problem.

2.1 Example: Binary Image Restoration

The earliest use of graph cuts for energy minimization in vision is due to Greig et.al. [14]. They consider the problem of restoration for a binary image. Given a binary image corrupted by noise, the task is to restore the original image. This is clearly simple binary optimization, with variables corresponding to image pixels, and terminals source and sink representing the two possible intensity values. Thus all the image pixels are taken for the nonterminal nodes of the graph. For concreteness, let us choose the source to represent intensity 0 and the sink to represent intensity 1. Let I be the observed image, and $I(p)$ be the observed intensity at pixel p. Let $D_p(l)$ be the penalty for assigning to pixel p an intensity $l \in \{0, 1\}$. Naturally, if we observe 0 at pixel p, then $D_p(0)$ should be smaller than $D_p(1)$, and vice versa. To encode these data constraints, for each pixel node p we create two t-links. The first t-link is (s, p) and it has weight $D_p(1)$. The second t-link is (p, t) and it has weight $D_p(0)$. Since we allow only nonnegative weights in the graph, we require that $D_p \geq 0$. This results in no loss of generality, see section 4.



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

Fig. 2. Example of a graph.Edge costs are reflected by their thickness. This graph construction was first used in Greig et. al. [14].

At this point, we have encoded all the constraints that come from the data, and we need to add the regularizing constraints to the problem. A reasonable assumption is that the original image is piecewise constant. To encode the piecewise constancy constraint, we create n-links between each pixel node and its 4 nearest neighbors in the image. We could use the 8 connected neighborhood, or any other desired neighborhood system instead. The weight of these n-links is set

to a smoothing parameter $\lambda > 0$. Thus the minimum cut is encouraged to sever as few n-links as possible, and the corresponding restored image is piecewise constant. The parameter λ weights the relative importance of the data constraints and the regularizing constraints. If λ is small, the data term is more important, and the optimum solution is piecewise constant to a smaller extent. If λ is large, the regularizing term is more important and the optimum solution is piecewise constant to a larger extent. Let us consider the two extremes of parameter λ. If $\lambda = 0$, then regularazing term is completely ignored and the optimum solution is just the observed image itself. If $\lambda = \infty$, the optimum consists of either all 0's of all 1's, depending on which intensity the majority of pixels prefer. Figure 2 shows the resulting graph and a its possible minimum cut. The thinner the edge, the smaller is its cost.

Let us calculate the cost of any cut C in this graph. First let us consider the contribution to the cost of C from the t-links. If pixel $p \in \mathcal{S}$, then the t-link (p, t) must be severed, and p is assigned intensity 0 in the restored image. Recall that the weight of (p, t) is $D_p(0)$, and this is exactly the penalty that we should pay for assigning intensity 0 to pixel p. Similarly, if pixel $p \in \mathcal{T}$, then p is assigned intensity 1, the t-link (s, p) is severed, and its weight is $D_p(1)$, the penalty for assigning intensity 1 to pixel p.

Now let us consider the cost that the n-links contribute to the cut. If pixels p and q are neighbors, then there are an n-links (p, q) and (q, p) in the graph. Suppose that p and q are on the same side of the cut, that is both are in $\mathcal{S}$ or both are in $\mathcal{T}$. Then the n-links (p,q) and (q,p) are not severed and thus do not contribute to the cost of the cut. However if f p and q are not on the same side of the cut (that is if one of them is in $\mathcal{S}$ and the other one is in $\mathcal{T}$), than theses n-links are severed and contribute their weight λ to the cost of the cut. Thus a cut on this graph has the cost that corresponds to the following energy function:

$$E(f) = c(\mathcal{S}, \mathcal{T}) = \sum_{p \in \mathcal{V}} D_p(f_p) + \sum_{(p,q) \in \mathcal{E}} \lambda \cdot \mathcal{I}(f_p \neq f_q), \tag{2}$$

where f_p is the binary variable at pixel p, f is the collection of such variables for all nonterminal nodes in $\mathcal{V}$, and, finally, $\mathcal{I}(\cdot)$ is the identity function which is 1 if it's argument is true and 0 otherwise. Stated simply, the first term says that the variable assignment f should agree with the observed data, and the second term states that most nearby pixels should have the same intensity label.

In general, we can use graph construction in figure 2 for any binary "labeling" problem of the following form. Suppose we have 2 labels which we wish to assign to a set of sites $\mathcal{P}$. Sites can be image pixels, medical volume voxels, and so on. Binary labels in the label set L can encode any property we may wish to assign to sites, for example intensity, object/figure membership, and so on. Suppose we know the penalty $D_p(l)$ which we should incur if pixel p is assigned to label $l \in L$, and we wish to find a piecewise constant labeling. Then the energy we may wish to optimize is exactly the one in equation 2, with the first sum over sites $\mathcal{P}$, the appropriate data terms, and the edges $\mathcal{E}$ expressing the neighborhood relations among the sites $\mathcal{P}$.

Previously, exact minimization of energies like (2) was not possible and such energies were approached mainly with iterative algorithms like simulated annealing [12]. 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. They did not think of this binary restoration problem as a serious application. Thus it is not entirely surprizing that the result of Greig et.al. remained unnoticed for almost 10 years mainly because the binary labeling limitation looked too restrictive.

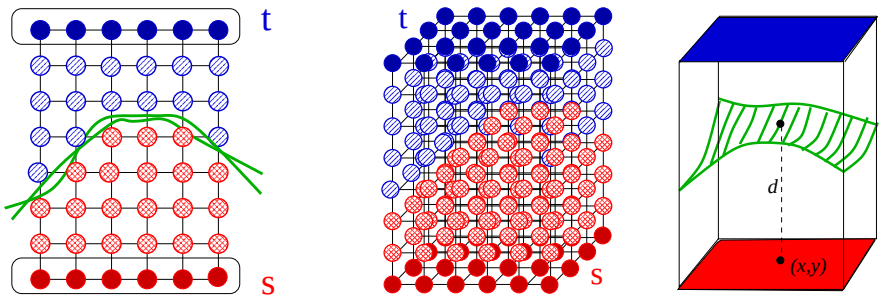
3 Graph Cuts as Hypersurfaces

Solution of many problems in vision, image processing and graphics can be represented in terms of optimal hypersurfaces. This section describes a geometric interpretation of graph-cuts as hypersurfaces in N-D manifolds that makes them an attractive framework for problems like image segmentation, restoration, stereo, photo/video editing, texture synthesis, and others.

We show a basic idea allowing to view s-t cuts as hypersurfaces, discuss interesting theories that make various connections between discrete graph cuts and hypersurfaces in continuous spaces, and provide a number of recently published examples where hypersurface view of graph cuts found interesting applications in computer vision, medical imaging, or graphics.

3.1 Basic idea

Consider two simple examples in Figure 3. Through out Section 3 we assume



(a) a cut on a 2D grid (b) a cut and a separating hypersurface in 3D

Fig. 3. s-t cut on a grid corresponds to binary partitioning of N-D space where the grid is imbedded. Such space partitioning may be visualized via a separating hypersurface. As shown in (a), multiple hypersurfaces may correspond to the same cut. However, such hypersurfaces become indistinguishable as the grid gets finer.

that a graph has no "soft" t-links, so that the source and the sink terminals are

directly connected only to some of the graph nodes via infinity cost t-links. In fact, all nodes hardwired to two terminals can be effectively treated as mutiple sources and multiple sinks that have to be separated by a cut. Figure 3 shows these sourses and sinks in dark red and dark blue colors. Such sources and sinks provide hard constraints or boundary conditions for graph cuts; any feasible cut must separate sources from sinks. Other nodes are connected to the sourses and sinks via n-links.

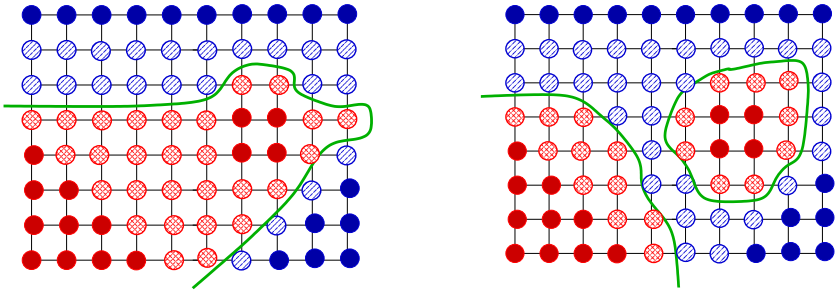
Without loss of generality (see Section 3.2), we can concentrate on feasible cuts that partition the simple 4- and 6- nearest neighbor grid-graphs in Figure 3 into two connected subsets of nodes: source component and sink component. Continuous 2D and 3D manifolds where the grid nodes are imbedded can be split into two disjoint contiguous regions containing the sinks and the sources, correspondingly. A boundary between two such regions are separating hypersurfaces shown in green color. As illustrated in Figure 3(a), there are many separating hypersurfaces that correspond to the same cut. They should all correctly separate the grid nodes of the source and the sink components, but they can "freely move" in the space between the grid nodes. Without getting into mathematical details, we will identify a class of all hypersurfaces corresponding to a given cut with a single hypersurface. In particular, we can choose a hypersurface that follows boundaries of "grid cells", or we can shoose "the smoothest" hypersurface. Note that the finer the grid, the harder it is to distinguish two separating hypersurfaces corresponding to the same cut.

Thus, any feasible cut on a grid in Figure 3 corresponds to a separating hypersurface in the imbedding continuous manifold. Obviously, the opposite is also true; any separating hypersurface corresponds to a unique feasible cut. Generalization of examples in Figure 3 would establish correspondance between $s-t$ graph-cuts and separating hypersurfaces in case of "fine" locally connected grids imbedded in N-D spaces. Following ideas in [6, 4], one can set a cost (or area) of each hypersurface based on the cost of the corresponding cut. This would impose a "cut metric" [4] in continuous N-D manifold imbedding the graph. By changing weights of n-links at grid nodes located in any particular point in space, one can tune local costs of all separating hypersurfaces that pass through such locations. Thus, "cut metric" can be tuned in specific applications to attract (repel) hypersyrfaces to (from) certain locations on N-D manifold. Then, standard combinatorial algorithms for computing minimum cost $s-t$ cuts (in Section 1.2) can be used as numerical tools for extracting globally optimal hypersurfaces.

3.2 Topological properties of graph cuts

The adjective "separating" implies that a hypersurface should satisfy certain hard constraints or boundary conditions; it shouls separate source and sink grid cells (seeds). Note that there are many freedoms in setting boundary conditions for graph cuts. Depending on hard constraints, topological properties of separating hypersurfaces corresponding to $s-t$ cuts may vary.

For example, we can show that the boundary conditions in Figure 3 guarantee that any feasible cut corresponds to topologically connected separating



(a) connected source segment (b) disjoint source segments

Fig. 4. Separating hypersurfaces can have different topological properties for the same set of hard constraints. Separating hypersurfaces in (a) and (b) correspond to two distinct feasible $s-t$ cuts. Min-cut/max-flow algorithms compute globally optimal hypersurface/cut without any restrictions on its topological properties as long as the sources and the sinks are separated.

hypersurface. For simplicity, we assume that our graphs are connected, that is, there are no "islands" of disconnected nodes. In Figure 3 all source and all sink nodes form two connected components. In such cases a minimum cost cut must partition the graph into exactly two connected subsets of nodes; one containing all sources and the other containing all sinks. Assuming that the minimum cost cut creates three or more connected components implies that some of these components contain neither sources, nor sinks. This contradicts minimality of the cut; linking any "no-source/no-sink" subset back to the graph corresponds to a smaller cost feasible cut.

Examples in Figure 4 illustrates different topological properties for separating hypersurfaces in more general cases where multiple disjoint components of sources and sinks (seeds) are present. Note that feasible $s-t$ cuts may produce topologically different separating hypersurfaces for the same set of boundary conditions.

In fact, controlling topological properties of separating hypersurfaces by setting up appropriate hard constraints is frequently a key technical aspect of applications using graph cuts. As discussed in Section 3.3, appropriate positioning of sourses and sinks is not the only tool to achieve desired topology. As shown in Figure 5, certain topological properties of separating hypersurfaces can be enforced via infinity cost n-links.

3.3 Applications of graph cuts as hypersurfaces

Below we consider several examples from recent publications where graph cuts are used as a method for extracting optimal hypersurfaces with desired topological properties.

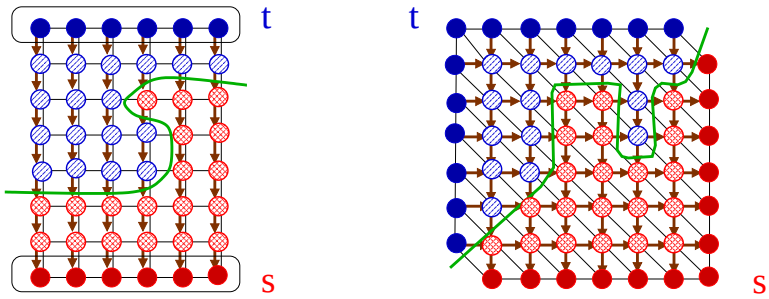
Methods for object extraction [6, 9, 25, 33] take full advantage of topological freedom of graph-cut based hypersurfaces. In particular, they allow to segment

objects of arbitrary topology. The basic idea is to set as sourses (red seeds) some image pixels that are known (a priori) to belong to an object of interest and to set as sinks (blue seeds) some pixels that are known to be in the background. A separating hypersurface should coinside with a desirable object boundary separating object (red) seeds from background (blue) seeds, as demonstrated in Figure 4. Cut metric can be set to reflect on image gradient. Pixels with high image gradients would imply low cost of local n-links and wice-versa. Then, minimal separating hypersurfaces tend to adhere to object boundaries with hight image gradients. Another practical strength of object extraction methods based on graph cuts is that they provide practical solutions for organ extraction problems in N-D medical image analysis [6]. One limitation of this approach to object extarction is that it may suffer from a bias to "small cuts" which could be often resolved with proper constraining of solution space.

Stereo was one of the first applications in computer vision where graph cuts were succesfully applied as a method for optimal hypersurface extraction. Two teams, Roy&Cox [28, 27] and Ishikawa&Geiger [16], almost simultaneously proposed two different formulations of stereo problem where disparity maps are interpreted as separating hypersurfaces on certain 3D manifolds. Their key technical contribution was to show that disparity maps (as optimal hypersurfaces) can be efficiently computed via graph cuts.

For example, Roy&Cox [28, 27] proposed a framework for stereo where disparity maps are separating hypersurfaces on 3D manidolds similar to one in Figure 3(b). Points of a bounded rectangular 3D manifold may be interpreted as points in 3D space observed by a pair of stereo cameras. The coordinate system can be chosen with respect to one of the cameras, so that a 3D scene point with coopdinates (x, y, d) would project to pixel (x, y) in that camera and d would be the depth of that point with respect to the same camera. Alternatively, if a pair of stereo images is rectified, then depth coordinate d can be re-interpreted as a "disparity", that is, a shift between pixel (x, y) in the first camera and pixel $(x + d, y)$ in the second camera where the same 3D scene point projects. A disparity map is a hypersurface $d = f(x, y)$ that can be viewed as a function assigning certain depth/disparity to each pixel (x, y). Note that hypersurface $d = f(x, y)$ separates the bottom and the top (facets) of 3D manifold in Figure 3(b). Thus, an optimal disparity map can be computed using graph cuts as an efficient discrete model for extracting minimal separating hypersurfaces.

As typical for applications using graph cuts, the cost (area) of each hypersurface is determined by a sum of weights of severed n-links in the imbedded graph. According to [28], weights of n-links at node (x, y, d) depend on color consistency between two cameras: if intensities of pixels (x, y) and $(x + d, y)$ in two cameras are similar then the likelihood that two pixels see the same 3D object point is high and the cost of n-links should be small. Later, [17, 27, 5] suggested anisotropic cut metric where vertical n-links are based on the same likelihoods as above but horizontal n-links are fixed to a constant that encorages smoother hypersurfaces avoiding unnecessary jumps between disparity levels.



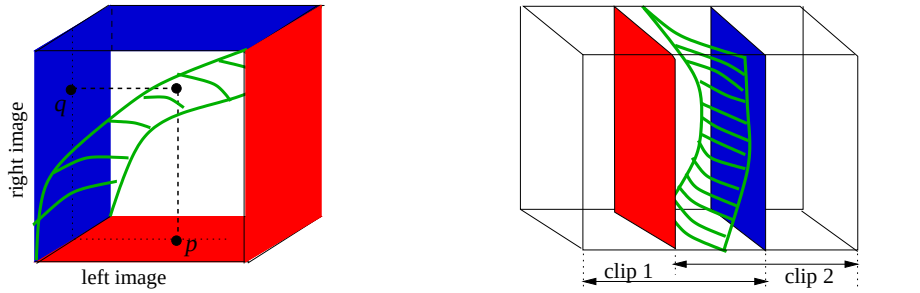
(a) Infeasible folding in [28, 27] (b) Infeasible folding in [16]

Fig. 5. Graph-cuts approach allows to impose certain additional topological constraints on separating hypersurfaces, if necessary. For example, [17, 5] proposed infinity cost directed n-links, shown in brown color in (a), that forbid folds on separating hypersurfaces in Figure 3. In particular, a hypersurface in Figure 3(b) without such folds corresponds to a diparity map $d = f(x, y)$ according to [28, 27]. Also, [16] impose monotonicity/ordering constraint on their disparity maps by adding infinity cost directed n-links (in brown color) that make illegal topological folds shown in (b). For clarity, examples in (a) and (b) correspond to single slices of 3D manifolds in Figure 3(b) and 6(a).

In general, separating hypersurfaces in Figure 3(b) can have folds that would make them inappropreate as disparity maps $d = f(x, y)$. If a minimum hypersurface computed via graph cuts has a fold then we did not find a feasible disparity map. Therefore, [17, 5] propose a set of hard constraints that make topological folds (see Figure 5(a)) prohibitively expensive. Note that additional infinity cost vertical n-links (directed down) make folds infeasible. This topological hard constraint take advantage of "directed" nature of graph cuts; a cost of a cut includes only severed directed edges that go from the (red) nodes in the source component to the (blue) nodes in the sink component. A cut with an illeagal fold in Figure 5(a) includes one infinity cost n-link.

Ishikawa&Geiger [16] also solve stereo via graph cuts by presenting disparity maps as separating hypersurfaces but they use different 3D manifolds and a different set of boundary conditions. They interprete a separating hypersurface $z = f(x, y)$ as a correspondence mapping between pixel $p = (x, y)$ in the left image and pixel $q = (f(x, y), y)$ in the right image (of a rectified stereo pair), as shown in Figure 6(a). Assignment of correspondences may be ambiguous if a hypersurface has folds like one in Figure 5(b). In order to avoid ambiguity, [16] introduce monotonicity (or ordering) constraint that is enforced by directed infinity cost n-links shown in brown color. Note that a cut in Figure 5(b) severs two brown n-links that go from a (red) node in a source component to a (blue) node in a sink component. Thus, the cost of the cut is infinity and the corresponding separating hypersurface with a fold becomes infeasible.

Similar to [28, 27], the cut metric on manifold in Figure 6(a) is based on color consistency constraint: a 3D points (x, y, z) on the manifold has low n-link costs



(a) Hypersurface as correspondence (b) Hypersurface separates two video clips

Fig. 6. Two more examples of graph cuts as separating hypersurfaces. Formulation of stereo problem in [16] computes pixel correspondences represented by a separating hypersurface on a 3D manifold in (a). A smooth transition between two video clips is performed in [23] via graph cuts computing globally optimal separating hypersurface in a 3D region of overlap between two clips in (b).

if intensity of pixel (x, y) in the left image is close to intensity of pixel (z, y) in the right image. Note that hyperplanes parallel to diagonal crossection (from bottom-left to top-right corners) of manifold in Figure 6(a) give correspondence mappings with constant stereo disparity/depth levels. Thus, spacial consistency of disparity/depth map can be enforced with anisotropic cut metric where diagonal n-links (from left-bottom to right-top corner) are set to a fixed constant representing penalty for jumps between disparity levels.

Another interesting example of graph-cuts/hypersurface framework is a method for video texture synthesis in [23]. The technique is based on computing a seamless transition between two video clips as illustrated in Figure 6(b). Two clips are overlaped in 3D (pixel-time) space creating a bounded rectangular manifold where transition takes place. A point in this manifold can be described by 3D coordinates (x, y, t) where (x, y) is a pixel position and t is time. The transition is represented by a separating hypersurface $t = f(x, y)$ that specifies for each pixel when to switch from clip 1 to clip 2. During transition a frame may have a mix of pixels from each clip. The method in [23] suggest a specific (cut) metric that for each point (x, y, t) in the ovelap region computes intensity difference between two clips. Small difference indicates a good moment (in space and time) for seamless transition between the clips and n-links at such (x, y, t) points are assigned a low cost. Note that "seamless transition" is a purely visual effect and it may be achieved with any separating hypersurface in Figure 6(b). In this case there is no real need to avoid hypersurfaces with "folds". A "fold" simply allows a pixel to switch between clip 1 and clip 2 a few times.

3.4 Theories connecting graph-cuts and hypersurfaces in R^n

In this section we briefly discuss a number of recent results demonstrating interesting theoretical connections between descrete graph cuts and hypersurfaces in

continuous spaces. It has been long argued that descrete algorithms on graphs, including graph cuts, may suffer from metrication artifacts. In particular, 4- and 6- nearest neighbor connections on 2D and 3D grids correspond to "Manhatten distance" metric that may produce blocky segments.

[4] and Kirsanov's paper [19]. Referencing Strang [29], and Ben Appleton is a good idea too.

4 Graph Cuts for Multi-Label Energy Minimization

In this section, we show that even though graph cuts provide an inherently binary optimization, they can be used for multi-label energy minimization. This minimization is sometimes exact, but usually it is approximate. There is an interesting connection between the exact multi-label optimization and the graph cuts hypersurface interpretation of section ???., which we explore in section 4.1. We begin by stating the general labeling problem, then in section 4.1 we describe the case when optimization can be performed exactly, and finally in section 4.2 we describe the approximate minimization approaches and their quality guarantees.

Many problems in vision (and indeed in other fields, like graphics) can be formulated as labeling problems. We have already seen an example of a labeling problem in section 2.1. We now state it in a more general form. In a labeling problem we have a set of sites $\mathcal{P}$, which can represent pixels, voxels, or any other set of entities. We also have a finite set of labels $\mathcal{L}$, which is now allowed to be of size larger than 2. Labels can represent any property that we wish to assign to sites, for example intensity, stereo disparity, a motion vector, and so on. The goal is to find a labeling f, which is a mapping from sites $\mathcal{P}$ to labels $\mathcal{L}$. As before, we use f_p to denote the label assigned to site p and f to denote the collection of such assignments for all sites in $\mathcal{P}$.

For each problem, we can derive a set of constraints which the optimal labeling f should obey as much as possible. Usually these constraints are derived from the observed data, and from prior knowledge. The data constraints are typically expressed as individual preferences of each site $p \in \mathcal{P}$ for labels in $\mathcal{L}$. To formalize them, for each pixel p, we use a function $D_p(l) : \mathcal{L} \rightarrow \mathcal{R}$. The smaller the value of $D_p(l)$, the more likely is the label l for site p. If for some p, $D_p^m = min_{l \in \mathcal{L}} D_p(l) < 0$, then we subtract D_p^m from $D_p(l)$ for all $l \in \mathcal{L}$. This does not change our energy formulation. Thus from now on, we assume, without loss of generality, that $D_p(l) \geq 0$ for all l, p.

The prior knowledge can in general be arbibrarily complex, but in graph cuts based optimization, we are esentially limited to constraints which impose different types of spatial smoothness on the labeling f. In order to enforce these smoothness constraints, we define a neighborhood system $\mathcal{N} \in \mathcal{P} \times \mathcal{P}$ on the sites $\mathcal{P}$. A neighborhood system contains pairs of pixels which are immediate neighbors. The simplest example of a neighborhood system is the one given by the 4 connected grid. We use a pairwise penalty function $V_{p,q}(l_p, l_q)$ which assigns a positive cost if neighboring pixels p and q are given different labels l_p and l_q. To enforce smoothness, $V_{p,q}(l_p, l_q)$ should be a nondecreasing function

of $||l_p - l_q||$ [1]. Different choices of $V_{p,q}(l_p, l_q)$ lead to different assumed types of smoothness. We discuss this issue in more detail in sections 4.1 and 4.2.

Now we are ready to formulate the labeling problem in terms of energy minimization. The optimal labeling f is the one which minimizes the following energy function:

$$E(f) = \sum_{p \in \mathcal{P}} D_p(f_p) + \sum_{(p,q) \in \mathcal{N}} V_{p,q}(f_p, f_q), \tag{3}$$

An energy functions of the type in 3 can be derived as the MAP estimate of a Markov Random Field ([8]), and hence is justified by Bayesian statistics.

One obvious limitation of the energy function in 3 is that in only depends on the pairwise interaction between pixels, and thus smoothness can only be encoded to the first degree. It is interesting to note that graph cuts can be used for energies with the terms depending on triples of pixels [22]. However the allowed form of these "triple cliques" is very limited, and at this point it is not clear if they can be used to encode a higher order smoothness on f.

4.1 Exact Optimization

In this section, we explore the case when the energy function in equation 3 can be optimized exactly with a minimum cut. We have to make the assumption that labels can be linearly ordered. This assumption limits the applicability of the method. For example, we cannot use it directly for motion estimation, since motion labels are 2 dimensional and cannot be linearly ordered.[2]. Without loss of generality, let us assume that labels are integers in the range $\mathcal{L} = \{1, ..., k\}$. Let $V_{p,q} = \lambda_{pq}|f_p - f_q|$. Then the energy function we seek to optimize is:

$$E(f) = \sum_{p \in \mathcal{P}} D_p(f_p) + \sum_{(p,q) \in \mathcal{N}} \lambda_{pq}|f_p - f_q|, \tag{4}$$

In vision, [16, 5] were the first to discover that the energy in equation 4 can be optimized exactly with a minimum cut on a certain graph $\mathcal{G}$, which we construct as follows. As usual, vertices $\mathcal{V}$ contain terminals s and t. For each site p, we create a set of nodes $p_1, ..., p_{k-1}$. We connect them by edges $\{t_1^p, ..., t_k^p\}$, where $t_1^p = (s, p_1)$, $t_j^p = (p_{j-1}, p_j)$, and $t_k^p = (p_{k-1}, t)$. Each edge t_j^p is assigned a weight $K_p + D_p(j)$, where $K_p = 1 + (k-1)\sum_{q \in \mathcal{N}_p} \lambda_{pq}$. Here $\mathcal{N}_p$ is the set of all neighbors of site p. For each pair of neighboring sites p, q and for each $j \in \{1, ..., k-1\}$ we create an edge (p_j, q_j) with weight $\lambda_{pq}/2$. Figure 7 illustrates the part of $\mathcal{G}$ which corresponds to two neighboring sites p and q in case when $|\mathcal{L}| = 4$.

For each site p, a cut on the graph $\mathcal{G}$ will sever at least one edge t_i^p. The weights for t_i^p are defined to be sufficiently large so that the minimum cut severs exactly one such edge for each site, see [5] for details. This establishes a natural correspondence between the minimum cut on $\mathcal{G}$ and an assignment of a label to

[1] Here we used the norm $||\cdot||$ notation because, in general, l_p may be a vector

[2] Iterative application of the algorithm described here was used for motion in [26]

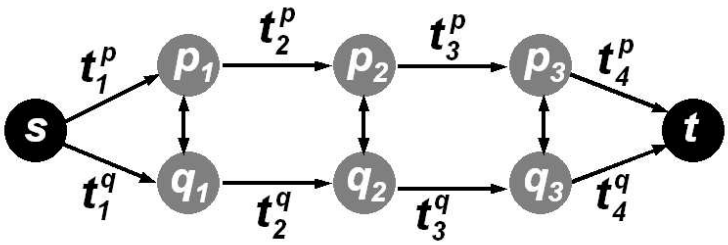


Fig. 7. Part of the graph construction for energy minimization in equation 4

p. If the minimum cut severs edge t_i^p, assign label i to p. The corresponding cost from all the links t_i^p to the cut is a constant plus $\sum_{p\in\mathcal{P}} D_p(f_p)$, that is the data term contribution to the energy.

Suppose that for neighboring sites p and q, the minimum cut severs t_a^p and t_b^q, and without loss of generality assume that $a < b$. Then it is easy to show that the minimum cut must also must sever all links (p_j, q_j) and (q_j, p_j) for $a \leq j < b$, see [5], and their total cost is $\lambda_{pq}|a-b|$. This cost is equal to the second term on the right hand side in equation 4. Thus the minimum cut gives the assignment of labels that minimizes the energy in equation 4.

Ishikawa [18] generalized the class of energy functions which can be minimized exactly with the minimum graph cut to the case when $V_{pq}(f_p, f_q)$ is a convex function. The construction is similar to the one given in this section, except even more edges between nodes p_i's and q_j's have to be added. Unfortunately, a convex V_{pq} is not suitable for a majority of vision applications if the number of labels in $\mathcal{L}$ is large. In vision, we usually seek a piecewise smooth labeling f, because object properties tend to be smooth everywhere except the object boundaries, where sharp jumps or *discontinuities* may be present. If the number of labels is large, then the label jump at a discontinuity, that is $|f_p - f_q|$, may be quite large. The penalty that a convex $V_{p,q}$ imposes on discontinuities is so large, that in the optimal f discontinuities are smoothed out with a "ramp". It is much cheaper to create a few small jumps in f rather than one large jump. Of all the convex $V_{p,q}$, the one in equation 4 is the most suitable for vision applications, since making a few small steps has the same cost as taking a one large step under the absolute difference measure. Nevertheless one sees oversmoothed disparity boundaries when using the energy in 4 in practice [30]. In essence, using a convex $V_{p,q}$ corresponds to the prior assumption that f is smooth everywhere.

It is interesting to note that the energy formulation in this section is equivalent to finding an optimal separating hyperplane approach of Roy and Cox [28]. In fact, in their first version of the algorithm [28] they only present the hypersurface interpretation of their algorithm. In a later version Roy [27] adds the energy minimization formulation.

4.2 Approximate Optimization

The pairwise penalty $V_{p,q}$ in the previous section is not discontinuity preserving because $V_{p,q}(f_p, f_q)$ is allowed to grow arbitrarily large as a function of $|f_p - f_q|$. One way to construct a discontinuity preserving $V_{p,q}$ is to cap its maximum value. Perhaps the simplest example of a discontinuity preserving $V_{p,q}$ is $V_{p,q} = \lambda_{pq} \cdot \mathcal{I}(f_p \neq f_q)$ [8]. This pairwise penalty is typically called the Potts model. We have already seen Potts $V_{p,q}$ in section 2.1[3], and it corresponds to the prior that the optimum labeling f is piecewise constant. Unfortunately, energy minimization with Potts $V_{p,q}$ is NP-hard [8], however graph cuts can be used to find an answer which is within a factor of 2 from the optimum [8].

In this section, we describe two methods for approximating the energy in 3, the expansion move and the swap move algorithms [8]. According to the results in [22], the swap move algorithm may be used whenever $V_{p,q}(\alpha,\alpha)+V_{p,q}(\beta,\beta) \leq V_{p,q}(\alpha,\beta) + V_{p,q}(\beta,\alpha)$ for all $\alpha, \beta \in \mathcal{L}$. In such case we say that $V_{p,q}$ satisfies the swap inequality. The expansion move algorithm can be used whenever $V_{p,q}(\alpha,\alpha) + V_{p,q}(\beta,\gamma) \leq V_{p,q}(\alpha,\gamma) + V_{p,q}(\beta,\alpha)$ for all $\alpha, \beta, \gamma \in \mathcal{L}$. In this case we say that $V_{p,q}$ satisfies the expansion inequality. Any $V_{p,q}$ which satisfies the expansion inequality also satisfies the swap inequality, which is easily checked by replacing γ by β . Thus the expansion inequality is more restrictive.

Both swap and expansion inequalities are satisfied by important cases of discontinuity preserving $V_{p,q}$'s. The truncated linear $V_{p,q}(\alpha,\beta) = min(T, ||\alpha - \beta||)$ satisfies the expansion inequality. The truncated quadratic $V_{p,q}(\alpha,\beta) = min(T, ||\alpha - \beta||^2)$ satisfies the swap inequality. Here T is a positive constant, which is the maximum penalty for a discontinuity. The truncated linear and truncated quadratic $V_{p,q}$ correspond to the piecewise smooth prior assumption on labeling f. Small deviations in labels between the nearby sites incur only a small penalty, thus the smoothness is encouraged. However sharp jumps in labels are occasionally permitted because the penalty T is not too severe to prohibit them.

Local Minimum with Respect to Expansion and Swap Moves Both the expansion and the swap algorithms find a local minimum of the energy function. However, in discrete optimization, we need to define what we mean by "a local minimum". For each labeling f, we define a set of moves M_f. Intuitively, these are the moves to other labelings that are allowed from f. Then we say that f is a local minimum with respect to the defined set of moves, if for any $f' \in M_f$, $E(f') \geq E(f)$. In most discrete optimization methods, for example, in simulated annealing [12] and ICM [2], they use what we call *standard* moves. These standard moves are defined as follows. Let $H(f, f')$ measure the number of sites for which f and f' differ, that is the Hamming distance between f and f'. Then for each f, the standard moves are $M_f = \{f' | H(f, f') \leq 1\}$. Thus a standard move allows to change a label of only one site in f, and $|M_f|$ is linear in the number of sites and the number of labels. It is easy to write an algorithm

[3] In the binary case, it is typically called the Ising model.

which finds a local minimum with respect to the standard moves. Start with an initial labeling f, cycle over all sites changing the label at each cite so that the total energy is decreased, until convergence to a local minimum. However the results are very dependent on the initial point, because a high dimensional energy in equation 3 has a huge number of local minima. Thus minimization with respect to standard moves produces a solution which can be arbitrarily far from the global minimum.

We now define the swap moves. Given a labeling f and a pair of labels α and β, a move $f^{\alpha\beta}$ is called an α-β swap if the only difference between f and $f^{\alpha\beta}$ is that some sites that were labeled α in f are now labeled β in $f^{\alpha\beta}$, and some sites that were labeled β in f are now labeled α in $f^{\alpha\beta}$. M_f is then defined as the collection of α-β swaps for all pairs of labels $\alpha, \beta \in \mathcal{L}$.

We now define the expansion moves. Given a labeling f and a label α, a move f^{α} is called an α-expansion if the only difference between f and f^{α} is that some sites that were not labeled α in f are now labeled α in f^{α}. M_f is then defined as the collection of α-expansions swaps for all labels $\alpha \in \mathcal{L}$. Figure 8 shows

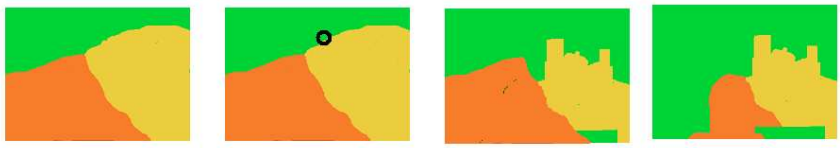


Fig. 8. From left to right: a labeling f, a labeling within one standard move of f (the changed site is highlighted by a black circle), labeling within one green-yellow swap of f, labeling within one green expansion of f.

an example of standard move versus α-expansion and α-β swap. Notice that a standard move is a special case of an α-expansion and a α-β swap. However there are α-expansion moves which are not α-β swaps and vice versa.

The expansion (swap) move algorithms find a local minimum with respect to expansion (swap) moves. The number of expansion (swap) moves from each labeling is exponential in the number of sites. Thus direct search for an optimal expansion (swap) move is not possible. This is where graph cuts are essential. It is possible to compute the optimal α-expansion or the optimal α-β swap with the minimum cut on a certain graph. This is because computing an optimal α-expansion (optimal α-β swap) is a binary minimization problem which happens to be regular [22] when the expansion (swap) inequality holds.

The criteria for a local minimum with respect to the expansions (swaps) are so strong that there are significantly fewer of such minima in high dimensional spaces compared to the standard moves. Thus the energy function at a local minimum is likely to be much lower. In fact, it can be shown that the local

minimum with respect to expansion moves is within a constant factor of optimum. The best approximation is in case of the Potts model, where this factor is 2. It is not surprizing then that most applications based on graph cuts use the expansion algorithm with the Potts model [5, 3, 20, 21, 31, 23, 24, 15, 1, 32].

Swap Move Algorithm In this section, we use the original construction [8] which only works in case $V_{p,q}(\alpha,\alpha) = 0$ for all $\alpha \in \mathcal{L}$. In this case, the graph construction is particularly simple, and is basically identical to the one in section 2.1. For the construction in the more general case (as given by the swap inequality), consult [22].

We start with an initial labeling f. We then cycle (in random order) until convergence over pairs of labels $\alpha, \beta \in \mathcal{L}$, at each cycle we find the optimal $f^{\alpha\beta}$ out of all α-β-swaps, and change current labeling to $f^{\alpha\beta}$. Obviously this cannot lead to an increase in energy, and at convergence we found the minimum with respect to the swap moves. Thus the key step is how to find the optimal α-β-swap. This step is performed by finding the minimum cut on a certain graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.

Let $\mathcal{V}$ contain the terminal source s, which we identify with α, and the sink terminal t, which we identify with β. Let $\mathcal{P}_{\alpha\beta}$ be the set of all sites for which either $f_p = \alpha$ of $f_p = \beta$. We include sites in $\mathcal{P}_{\alpha\beta}$ as nonterminal nodes in $\mathcal{V}$. Each nonterminal node is connected to the source with a t-link of weight $D_p(\beta) + \sum_{q \in \mathcal{N}_p, q \notin \mathcal{P}_{\alpha\beta}} V(\beta, f_q)$, and to the sink with a t-link of weight $D_p(\alpha) + \sum_{q \in \mathcal{N}_p, q \notin \mathcal{P}_{\alpha\beta}} V(\alpha, f_q)$. For each pair of neighboring nodes p, q in $\mathcal{P}_{\alpha\beta}$ we create an n-link (q, p) with weight $V_{p,q}(\alpha, \beta)$ and an n-link (p, q) with weight $V_{p,q}(\beta, \alpha)$. We make a correspondence between cuts $C = (\mathcal{S}, \mathcal{T})$ and moves which α-β-swaps as follows. If $p \in \mathcal{S}$, we set $f_p^{\alpha\beta} = \alpha$, and if $p \in \mathcal{T}$ we set $f_p^{\alpha\beta} = \beta$. If p was not a part of $\mathcal{G}$ (that is if $f_p \neq \alpha, \beta$), then we set $f_p^{\alpha\beta} = f_p$. Obviously any cut corresponds to an α-β swap, and we can show that the minimum cut corresponds to the optimum α-β swap in the same way as in section 2.1. [8].

Expansion Move Algorithm We start with aninitial labeling f. We then cycle (in random order) until convergence over all labels $\alpha \in \mathcal{L}$, find the optimal f^α out of all α-expansions, and change current labeling to f^α. Obviously this cannot lead to an increase in energy, and at convergence we found the local minimum with respect to expansion moves. Thus the key step is how to find the optimal α-expansion, which is performed by finding a minimum cut on a certain graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.

For the expansion move algorithm we basically follow the construction in [22] because it leads to a simpler graph. Let $\mathcal{V}$ contain the terminal source s, which we identify with α, and the sink terminal t, which we identify with $\bar{\alpha}$. We include all sites in $\mathcal{P}$ as nonterminal nodes in $\mathcal{V}$. Each nonterminal node p is connected to the source with a t-link of the weight $D_p(f_p)$, and to the sink with a t-link of weight $D_p(\alpha)$. For each pair of neighboring nodes p, q we create an n-link (p, q) with weight $V_{p,q}(\alpha, f_q) + V_{p,q}(f_p, \alpha) - V_{p,q}(\alpha, \alpha) - V_{p,q}(f_p, f_q)$.

If $V_{p,q}(f_p, \alpha) - V_{p,q}(\alpha, \alpha) \geq 0$, then we add an additional weight $V_{p,q}(f_p, \alpha) - V_{p,q}(\alpha, \alpha)$ to the t-link (s, p), else we an additional weight $V_{p,q}(\alpha, \alpha) - V_{p,q}(f_p, \alpha)$ to the t-link (p, t). If $V_{p,q}(f_p, f_q) - V_{p,q}(f_p, \alpha) \geq 0$, then we add an additional weight $V_{p,q}(f_p, f_q) - V_{p,q}(f_p, \alpha)$ to the t-link (s, q), else we an additional weight $V_{p,q}(f_p, \alpha) - V_{p,q}(f_p, f_q)$ to the t-link (q, t).

We make a correspondence between cuts $C = (\mathcal{S}, \mathcal{T})$ and moves which α-swaps as follows. If $p \in \mathcal{S}$, we set $f_p^{\alpha\beta} = \alpha$, and if $p \in \mathcal{T}$ we set $f_p^{\alpha\beta} = f_p$. Obviously any cut corresponds to an α swap, and it can be shown that the minimum cut corresponds to the optimum α -expansion fairly close to the proof in section 2.1. [8].

References

1. M. Agrawal and L.S. Davis. Window-based, discontinuity preserving stereo. In *CVPR04*, pages I: 66–73, 2004.
2. J. Besag. On the statistical analysis of dirty pictures (with discussion). *Journal of the Royal Statistical Society, Series B*, 48(3):259–302, 1986.
3. S. Birchfield and C. Tomasi. Multiway cut for stereo and motion with slanted surfaces. In *ICCV99*, pages 489–495, 1999.
4. Y. Boykov and V. Kolmogorov. Computing geodesics and minimal surfaces via graph cuts. In *International Conference on Computer Vision*, volume I, pages 26–33, 2003.
5. 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.
6. Yuri Boykov and Gareth Funka-Lea. Optimal object extraction via constrained graph-cuts. *International Journal of Computer Vision (IJCV)*, 2005, to appear. (Earlier version is in ICCV'01, vol. I, pp. 105-112, July 2001).
7. Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, September 2004.
8. 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.
9. 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.
10. William J. Cook, William H. Cunningham, William R. Pulleyblank, and Alexander Schrijver. *Combinatorial Optimization*. John Wiley & Sons, 1998.
11. L. Ford and D. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.
12. S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.
13. Andrew V. Goldberg and Robert E. Tarjan. A new approach to the maximum-flow problem. *Journal of the Association for Computing Machinery*, 35(4):921–940, October 1988.

14. 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.
15. L. Hong and G. Chen. Segment-based stereo matching using graph cuts. In *CVPR04*, pages I: 74–81, 2004.
16. H. Ishikawa and D. Geiger. Occlusions, discontinuities, and epipolar lines in stereo. In *5th European Conference on Computer Vision*, pages 232–248, 1998.
17. H. Ishikawa and D. Geiger. Segmentation by grouping junctions. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 125–131, 1998.
18. Hiroshi Ishikawa. Exact optimization for Markov Random Fields with convex priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(10):1333–1336, 2003.
19. Danil Kirsanov and Steven J. Gortler. A discrete global minimization algorithm for continuous variational problems. *Harvard Computer Science Technical Report*, TR-14-04, July 2004.
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. Vladimir Kolmogorov and Ramin Zabih. What energy functions can be minimized via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):147–159, February 2004.
23. Vivek Kwatra, Arno Schodl, Irfan Essa, and Aaron Bobick. GraphCut textures: image and video synthesis using graph cuts. In *ACM Transactions on Graphics (SIGGRAPH)*, volume 22, July 2003.
24. M.H. Lin and C. Tomasi. Surfaces with occlusions from layered stereo. *PAMI*, 26(8):1073–1078, August 2004.
25. Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. Grabcut - interactive foreground extraction using iterated graph cuts. In *ACM Transactions on Graphics (SIGGRAPH)*, August 2004.
26. Sbastien Roy and Venu Govindu. MRF solutions for probabilistic optical flow formulations. In *International Conference on Pattern Recognition (ICPR)*, September 2000.
27. Sebastien Roy. Stereo without epipolar lines: A maximum-flow formulation. *International Journal of Computer Vision*, 34(2/3):147–162, August 1999.
28. 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.
29. Gilbert Strang. Maximal flow through a domain. *Mathematical Programming*, 26:123–143, 1983.
30. Olga Veksler. *Efficient Graph-based Energy Minimization Methods in Computer Vision*. PhD thesis, Cornell University, August 1999.
31. J. Wills, S. Agarwal, and S. Belongie. What went where. In *CVPR03*, pages I: 37–44, 2003.
32. J. Xiao and M. Shah. Motion layer extraction in the presence of occlusion using graph cut. In *CVPR04*, pages II: 972–979, 2004.
33. Ning Xu, Ravi Bansal, and Narendra Ahuja. Object segmentation using graph cuts based active contours. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume II, pages 46–53, 2003.