Graph Cut Algorithms in Vision, Graphics and Machine Learning
An Integrative Paper

Sudipta N. Sinha { ssinha@cs.unc.edu}
University of North Carolina at Chapel Hill.

Abstract

This integrative paper studies graph-cut and network flow algorithms on graphs and compares its applications towards solving diverse problems in Computer Vision, Computer Graphics and Machine Learning. The following three papers form the core of this comparative study.

- **'An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision'** *by Boykov et.al.[1] reformulates a variety of problems in computer vision namely image restoration, stereo and segmentation into energy minimization problems and solves them using graph-cut algorithms.*
- **'Graph Cut Textures: Image and Video Synthesis Using Graph Cuts'** *by Kwatra et. al. [4] proposes a new algorithm for image and video texture synthesis in the area of Computer Graphics. At the core of their method is a graph-cut technique for computing optimal seams across multiple image patches which are to be blended and merged together to generate the synthesized textures.*
- **'Semi Supervised Learning using Randomized Mincuts'** *by Blum et.al. [5] proposes a graph mincut based approach to the machine learning problem of combining labeled and unlabeled data to do semi-supervised learning.*

1. Introduction

Network flow algorithms studied in the field of combinatorial optimization find widespread applications in optimization problems that can be represented by graphs containing nodes and arcs between these nodes. Although network flows are used to solve problems in physical networks, often a large variety of problems which have no inherent physical network can also be modeled using a network and a notion of flow in such a network. For instance the flow of liquid in pipes, bytes on a communication network or vehicles on highways can be modeled as flow networks. In such flow networks, one often needs to compute the maximum rate at which material flows in the flow network, (the maximum flow problem). Related to it is the question of cuts in flow networks, ie. the problem of determining a set of edges of total minimum capacity, which if removed will disrupt the flow in the network. The max-flow and min-cut problem will be studied and its applications to three different problems in Computer Vision, Graphics and Machine Learning will be described and compared in this paper. In each case, the original problem is transformed into a graph-cut problem which is in turn solved by computing the maximum flow on flow networks.

The rest of this paper is organized as follows. In Section 2, the background theory of the Max-flow problem is described along with an overview of the algorithms used to solve it. The theory of Markov Random Fields is presented in the context of the problems we study. Next in Section 3, 4 and 5, the specific problems are described ie. how each of them are reformulated as graph-cut problems and solved using max-flow algorithms. Section 6 investigates the motivation for using the graph-cut approach and Section 7 compares the three graph cut approaches. Finally conclusions and the effectiveness of the graph cut approach in the respective problem domain are discussed in Section 8.

2. Background

A *flow network* $G(V, E)$ is formally defined as a fully connected directed graph where each edge $(u, v) \epsilon E$ has a positive capacity $c(u, v) \geq 0$. Two special vertices in a flow network are designated the *source* s and the *sink* t respectively. A *flow* in G is a real-valued function $f : VXV \rightarrow R$ that satisfies the following three properties:

- Capacity Constraint:
  For all $u, v \epsilon V$, $f(u, v) \leq c(u, v)$.
- Skew Symmetry:
  For all $u, v \epsilon V$, $f(u, v) = -f(v, u)$.
- Flow Conservation:
  For all $u \epsilon (V\text{-}\{s, t\})$, $\sum_{v \epsilon V} f(u, v) = 0$.

The value of a flow is defined as $|f| = \sum_{v \epsilon V} f(s, v)$ ie. the total flow out of the source in the flow network G.

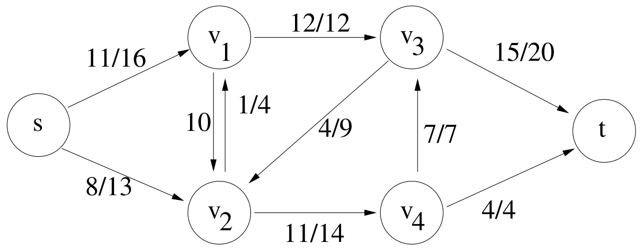


Figure 1: (a) The figure (taken from Cormen et. al. [10] shows a flow network $G(V, E)$ with a valid flow f. The values on the edges are $f(u, v)/c(u, v)$. The current flow has value 19, it is not a maximum flow.

2.1. The Max-Flow and Min-Cut Problem

The max-flow problem is to find the flow of maximum value on a flow network G. A *s-t cut* or simply *cut* of a flow network G is a partition of V into S and $T = V - S$ such that $s \epsilon S$ and $t \epsilon T$. For a given flow f, the *net flow* across the cut (S,T) is defined as $f(S, T) = \sum_{x \epsilon S} \sum_{y \epsilon T} f(x, y)$. Using a similar notation the capacity of a cut (S, T) is defined as $c(S, T) = \sum_{x \epsilon S} \sum_{y \epsilon T} c(x, y)$. A *minimum cut* of a flow network is a cut whose capacity is the least over all the s-t cuts of the network.

An example of a flow network with a valid flow is shown in Figure 1.

Theorem 1 *The max-flow min-cut theorem : If f is a flow in a flow network $G = (V, E)$ with source s and sink t then the value of the maximum flow is equal to the capacity of a minimum cut. Refer to Cormen et. al. [10] for the proof.*

The intuition behind the proof is as follows. The maximum flow must saturate edges in the flow network such that no further flow can be pushed. These saturated edges must lie on one of the min-cuts. This result allows one to compute the minimum cut of a flow network by first solving for the max-flow, for which polynomial time algorithms exist.

The single-source single-sink max-flow problem described above is a specific case of the more general multiway cut problem where there are k terminals and a multiway cut is a minimum set of edges which separates each terminal from all the others. It has been shown that if $k \geq 3$, the problem is NP-Hard. Some of the graph cut applications that we shall investigate in this paper will indeed require approximation algorithms for the multiway cut problem.

2.2. Max-Flow and Min-Cut Algorithms

The polynomial algorithms for the single-source single-sink max-flow problem can be divided into two classes, algorithms based on the Ford Fulkerson method [8] and those based on the "push-relabel" method [7]. The two contrasting approaches are described below. An approximation algorithm for the multiway cut problem will be described later in the context of the chosen problems, in Section 3.6.

The intuitive idea behind the Ford-Fulkerson method is that starting with zero flow ie. $f(u, v) = 0$ for all $u, v \epsilon V$, the flow can be gradually increased by finding a path from s to t along which more flow can be sent. Such a path is called an augmenting path, and once it has been found, the flow can be augmented along this path. The process if repeated, must end after a finite number of iterations after which no augmenting paths between s and t exist anymore. A typical algorithm of this type maintains for a given flow f, the residual graph of G, called G_f whose topology is identical to G but whose edge capacities stores the residual capacity of all the edges, given that there is already some flow in them. The search for an augmenting path at the i^{th} iteration is done on the current residual graph G_{f_i}. Once an augmenting path is found, the maximum amount of flow that can be sent down it, f_{incr} must saturate at least one of the edges of this augmented path. The new flow at the end of the iteration will be $f_i + f_{incr}$.

The running time complexity of different algorithms will in general vary depending on how the augmenting path is chosen. Dinic algorithm [9] that uses breadth-first search to find the shortest paths from s to t on the residual graph, has an overall worst case running time of $O(n^2 m)$, n being the number of nodes and m being the number of edges.

In contrast to the Ford-Fulkerson method where augmenting the flow operates on the complete residual graph, the Push-Relabel algorithms operate locally on a vertex at a time, inspecting only its neighbours. Unlike the Ford-Fulkerson method, the flow conservation property is not satisfied during the algorithm's execution. The intuitive idea here is to associate a notion of height along with all the nodes in the network. The height of the source and sink are fixed at $|V|$ and 0 respectively while at the start all other vertices are at height 0. The algorithm starts by sending flow down from the source and the amount of flow sent, saturates all the outgoing edges. All intermediate nodes have a buffer or a reservoir that can store excess flow. Nodes with positive excess flow are said to be overflowing nodes. Overflowing nodes try to push the excess flow downhill. However when an overflowing node finds the edges to its neighbours at the same height as itself saturated, it increments its own height, a process which is called "relabeling". This allows it to get rid of the excess flow. The algorithm terminates when none of the nodes in V are overflowing. Often excess flow accumulated in the interior nodes are sent back to the source by relabeling these nodes with height beyond $|V|$.

The generic push-relabel algorithms thus have two basic operations - "push" flow and "relabel" an overflowing node and Cormen et. al. proves that a generic push-relabel style algorithm has a $O(n^2 m)$ worst case running time, and there are certain $O(n^3)$ algorithms in this class. [7, 8] provide details on these various algorithms, data structures chosen and practical trade-offs encountered in actual max-flow implementations.

2.3. Markov Random Fields

This section briefly introduces the theory of Markov Random Fields (MRF) as it will be relevant in understanding the common thread between the three papers chosen for study. Markov Random Fields is a generative model often used in Image Processing and Computer Vision to solve labeling problems. A Markov Random Field consists of three sets, a set S of sites, a neighbourhood system N and a set (also called field) of ramdom variables F. The neighbourhood system $N = \{N_i \mid i \epsilon S\}$ where each N_i is a subset of sites of S which form the neighbourhood of site i. The random field $F = \{F_i \mid i \epsilon S\}$ consists of random variables F_i that take on a value f_i from a set of lables $L = \{l_1, l_2, \ldots\}$. A particular set of labels, often denoted by f (which can be thought of as the joint event $\{F_1 = f_1, F_2 = f_2, \ldots\}$) is called a configuration of F. The probability of a particular configuration f ie. $P(F = f)$ must satisfy the Markov property in order for F to be a Markov Random Field.

$$P(f_i|f_{S-i}) = P(f_i|f_{N_i}), \ \forall\ i \,\epsilon\, S.$$

This means that the state of each random variable F_i depends on the state of its neighbours, ie. $F_{N_i} = \{F_i| \, i \,\epsilon\, N_i\}$. However it has also been shown that the probability of a particular configuration is proportional to the sum of *clique potential* V_C over all the cliques in N. The *clique potential* is obtained from prior probabilities of a particular labeling of the sites in the clique C.

Markov Random Fields are used to model labeling problems where an optimal labeling is desired. From a probabilistic perpective, one wishes to estimate the configuration f based on observed data D (could be noisy or incomplete) that maximises the likelihood function, $P(D|f)$. Using Bayes Theorem, this likelihood function can be expressed as an energy function $E(f)$ and the maximum a posterieri (MAP) estimate of f should maximize this energy function.

Different MRF's differ in a choice of the neighbourhood system and the prior probabilities. In vision and image processing, where S, the set of sites often coincides with the set of regular grid of pixels and voxels, 4-neighbourhood or 8-neighbourhood systems on a 2D grid or 6-neighbourhood or 26-neighbourhood systems on a 3D grid are common. Morever, often a labeling with the following properties is desired; it should be locally constant or smooth but should also allow for discontinuities (at region boundaries). When one chooses clique potentials to make the desired labeling piecewise continuous, the resulting MRF is called Generalized Potts Model MRF (GPM-MRF). This formulation is useful for Vision and will be discussed in the next section. For more details on MRF's, refer to [11].

The problem of texture synthesis in graphics [4] can also be modeled as a MRF where sites of the MRF would typically be pixels or group of pixels in the output texture and the clique potentials in the MRF would depend on the similarity of pixel neighbourhoods in the input texture. The goal here is to generate an output texture perceptually similar to the input texture. Perceptual Similarity is essentially a labeling and grouping problem and is hence well modeled by MRFs.

Semi-Supervised Learning [5] is another labeling problem where the set of training examples form the set S, and similarity between each pair of examples in the training set is modeled using a neighbourhood system. The goal is to obtain a labeling or classification which intuitively separates dissimilar examples and clusters similar examples.

3. Energy Minimization Problems in Computer Vision

3.1. Problem Statement and Goal

Boykov et. al. [1] solve various computer vision problems which can be posed as energy minimization problems. Some examples of such problems are image restoration, stereo, multiple view reconstruction, and object segmentation. As explained in the previous section, these tasks can all be classified as labeling problems where visual constraints are typically contextual and reflect a choice of priors in the neighbourhood systems. Finding the most likely labeling translates to optimizing an energy function. The authors solve this energy minimization problem by transforming it into a multi-way cut problem on a graph. However the multi-way cut problem is NP-Hard and they propose an approximation algorithm that produces a cut which minimises the original energy function in a strong sense.

The generic energy minimization framework for labeling problems is now described followed by the specific examples of image restoration, stereo and object segmentation. We conclude this section by a description of the approximation algorithm which uses max-flow iteratively to solve the multi-way cut problem.

3.2. The Energy Minimization Framework

As described in Section 2.3, a large number of computer vision problems try to assign labels (such as intensity, disparity, segmentation regions) to pixels based on noisy measurements. In the presence of uncertainties, finding the best labeling becomes an optimization problem. Certains visual constraints are often used to constrain these labels. In vision and image processing, these labels often tend to vary smoothly within the image, except at some kind of region boundaries where discontinuities are allowed. The fact that a particular pixel label depends on the labels of its neighbours allows modeling the optimization problem as a MRF. Boykov et.al. [2] show that finding the most likely labeling for some given data, is equivalent to seeking the MAP (maximum a posteriori) estimate. In order to solve a particular vision problem, a suitable neighbourhood system needs to be chosen and prior probabilities for particular labelings in the neighbourhood system need to be assigned. Different choices for these entities yield different classes of energy functions and hence the term energy model is used to refer to this problem domain specific choice. Two common energy models are described below.

- Potts Interaction Energy Model

$$E(I) = \sum_{p \epsilon P} |I_p - I_p^o| + \sum_{(p,q) \epsilon N} K_{(p,q)}.T(I_p \neq I_q) \quad (1)$$

  where $I = \{I_p | \, p \, \epsilon \, P\}$ are the unknown true labels over the set of pixels P and $I^o = \{I_p^o | \, p \, \epsilon \, P\}$ are the observed labels corrupted by noise. The Potts interaction are specified by $K(p, q)$, the penalties for label discontinuities between adjacent pixels. The function $T()$ is an indicator function. This is a good model where the labels are likely to be piecewise constant with discontinuities at boundaries. The Pott energy can be optimally solved for binary labeling using max-flow, however the multiple label case is NP-Hard.

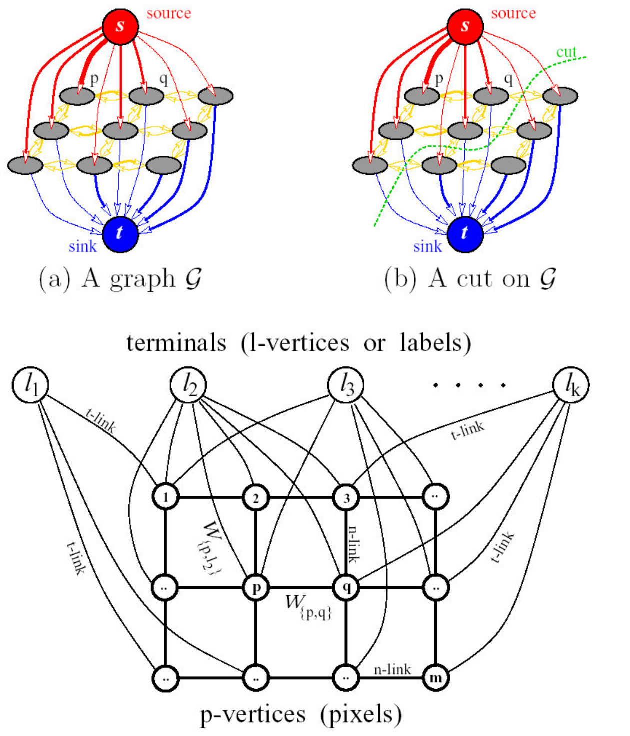
- Linear Interaction Energy Model

$$E(I) = \sum_{p \epsilon P} |I_p - I_p^o| + \sum_{(p,q) \epsilon N} A_{(p,q)}.T(I_p \neq I_q) \quad (2)$$

  where constants $A_{(p,q)}$ store the importance of the interaction between neighbouring pixels p and q. In contrast to the Pott Energy Model, the Linear Interaction Energy produces labelings which are piecewise smooth but with discontinuities at boundaries.

Figure 2 (taken from [2] is an example of a graph constructed from the Pott Energy Model. The graph G contain two kinds of vertices: *p-vertices* (pixels or voxels which are the sites in the associated MRF) and *l-vertices* (which coincide with the labels and will be terminals in the graph cut problem). All the edges present in the neighbourhood system N are edges in G. These edges are called *n-links*. Edges between the *p-vertices* and the *l-vertices* called *t-links* are added to the graph. *t-links* are assigned weights based on the data term (first term in Equations 1, 2) while *n-links* are assigned weights based on the interaction term (second term in Equation 1, 2). While *n-links* are bi-directional, *t-links* are uni-directional, leaving the source and entering the sink. Figure 2 shows the graph constructed for binary labeling on a small 3x3 pixel image.

In the multiple label case, the multiway cut should leave each *p-vertex* connected to one *l-vertex*. This ensures that



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

Figure 2: All figures were taken from Boykov et. al. [1]. (a) The figure shows a graph $G(V, E)$ constructed for binary labeling problem for a 3x3 pixel image on the left. (b) A cut along with the corresponding labeling is shown on the right. (c) The figure shows a graph $G(V, E)$ constructed from the energy function in Equation 1. The topology of the graph is automatically determined by the energy function and different problem that use the same energy model would result in an identical graph construction.

every multi-way cut which separates all terminals, must correspond to a valid labeling ie. a configuration of the associated MRF. The minimum cost multiway-cut will minimize the energy function in Equations 1, 2 where the severed *n-links* would correspond to the boundaries of the labeled regions. The approximation algorithm which finds this multi-way cut, is called the α-expansion algorithm [2]. It involves iteratively executing max-flow considering a particular label, (the α label) one at a time and is discussed in Section 3.6.

3.3. Image Restoration

The image restoration problem aims at recovering the original pixel intensities of an image when the observed image is noisy. Here the labels are the image intensities and the most likely labeling is obtained by minimizing an energy function similar to the ones described in Section 3.2. The visual

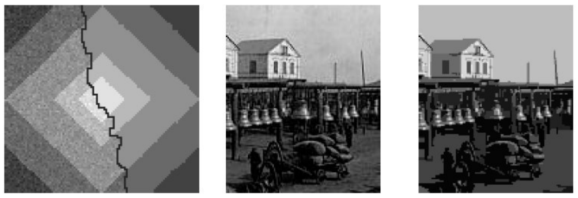

(a) *Diamond* restoration (b) Original *Bell Quad* (c) "Restored" *Bell Quad*

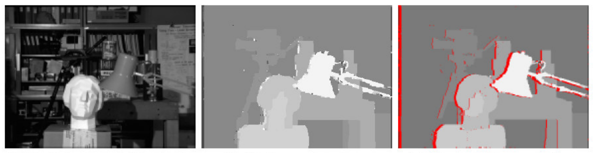
Figure 3: Image Restoration Examples taken from Boykov et. al. [1]. (a) Synthetic data : An example of a noisy image and its restored version. (b) & (c) Real data; An example of a noisy image and its restored version.

constraints exploited here are the fact that image intensities tend to vary smoothly in most images except at boundaries. Both the Pott Energy as well as Linear Interaction Energy model yields reasonable results. The actual choice of $K_{(p,q)}$ and $A_{(p,q)}$ determines the degree of smoothness in the restored images. Figure 3 shows examples from [1].

3.4. Stereo

Dense stereo is a popular method for 3D Reconstruction from two calibrated views of a scene. It involves first recovering matching pixels (pixels corresponding to the same 3D feature) in the two views and then recovering the depth of the 3D point by triangulation (intersecting rays backprojected from the two matching pixels). Finding accurate matching pairs for all pixels is a difficult problem to solve accurately because often such matching can be ambiguous depending on factors like camera baseline, amount of texture in the scene or the degree of specularity of objects in the scene. In a stereo pair, matching pixels are recovered using disparities. Every pixel $p_1(i,j)$ in the first image has a particular disparity d with respect to the matching pixel $p_2(i+d,j)$. A method called image rectification can ensure that corresponding pixels are always on identical scanlines in the rectified image pair. The problem of recovering an accurate disparity image can be posed as an energy minimization problem using the same MRF framework we have been studying. The problem of stereo is identical to the image restoration problem except that here the labels are disparity values. Disparity in a stereo pair tends to vary within a fixed range and can be discretely sampled. It also tends to vary smoothly over the image except at depth-discontinuities. Energy models like the Pott Energy, shown in Equation 1 can incorporate such contextual information within the MRF framework.

The stereo problem is harder compared to image restoration because of the presence of occlusions. Occlusion occur when 3D points are visible in only one of the stereo image pairs and are typically found near depth-discontinuities


(a) Left image of *Head* pair (b) Potts model stereo (c) Stereo with occlusions
`Disparity maps obtained for the` ***`Head`*** `pair`

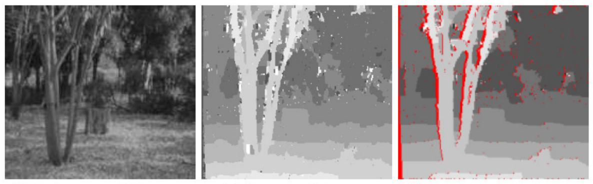

(d) Left image of *Tree* pair (e) Potts model stereo (f) Stereo with occlusions
`Disparity maps obtained for the` ***`Tree`*** `pair`

Figure 4: Stereo examples taken from Boykov et. al. [1]. One of the images in a stereo pair alongwith the computed disparity image. Top Row : Tsukuba dataset. Bottom Row : Tree dataset.

in the scene. Occlusions which make the visual correspondence problem harder, can be explicitly modeled in the Energy Minimization framework by a modified labeling problem of the following type. Sites in the MRF for this modified problem do not represent image pixels but pair of pixels which can potentially correspond. The set of labels is $\{0,1\}$ where 0 indicates either of the pixels are occluded and 1 indicates that the pair of pixels are matching. The new energy function is :

$$E(f) = E_{data}(f) + E_{occ}(f) + E_s(f)$$

where

$$E_{data} = \sum_{l(p,q)=1} D_{(p,q)}$$

is the term which imposes a penalty based on intensity differences of matching pixels p and q.

$$E_s = \sum_{\{(p,q),(p',q')\}\epsilon N} K_{\{(p,q),(p',q')\}}.T(l_{(p,q)} \neq l_{(p\prime,q\prime)})$$

is the smoothness term which forces adjacent pixels to have the same or relatively close disparities.

$$E_{occ} = \sum_{p\epsilon P_1 \bigcup P_2} C_p.T(p\ is\ occluded)$$

is the new term, the occlusion penalty term which imposes a penalty for making a particular pixel p in the stereo image pair P_1 or P_2 occluded. $T()$ is the indicator function in the above formulation. Minimizing the energy function is still NP-Hard but an approximate algorithm based on α-expansion computes a local minimum within a constant factor of the global minimum by solving max-flow on an associated graph.

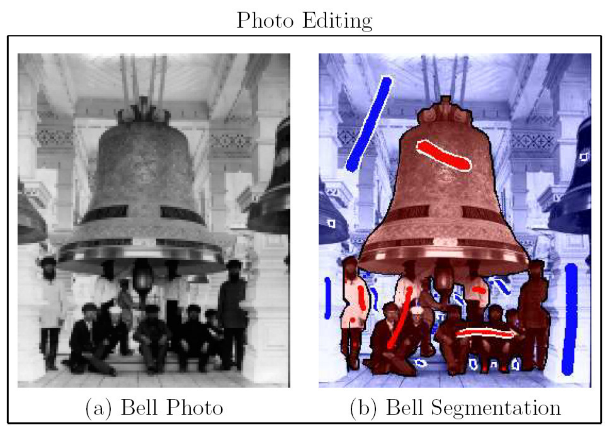


Figure 5: Image Segmentation examples taken from Boykov et. al. [1]. Interactive user input is used to guide the segmentation.

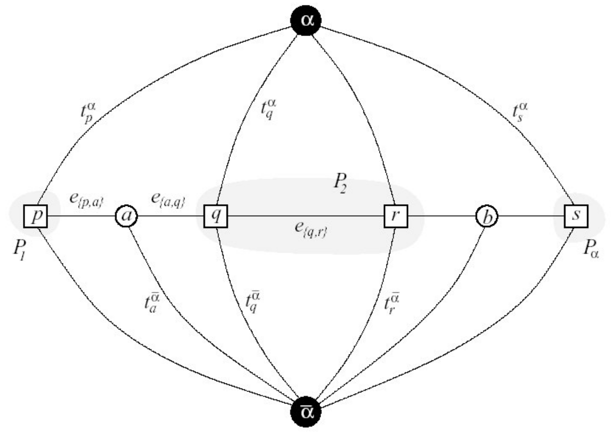
3.5. Segmentation

The Pott Energy function, (see Equation 1) comes up again in the context of image segmentation where the goal is to group image pixels into logical groups or segments which may represent objects in the scene. In 3D Segmentation, the grouping is done on voxels in volumetric data as is typically encountered in medical imaging. Segmentation is typically posed as a binary labeling problem where $\{$*foreground, background*$\}$ constitutes the set of labels typically assigned to pixels or voxels. The binary labeling problem for the Pott Energy function as mentioned earlier can be optimally solved by a single execution of max-flow. The graph construction and max-flow formulation is quite identical to the one for the image restoration problem (refer Section 3.3. To get accurate segmentations, user input is provided into the labeling problem by allowing the user to pre-label (these labels are not allowed to change) some pixels as foreground and some as background. This is illustrated in Figure 5.

3.6. The α-expansion algorithm

Boykov et. al. [2] proposes a fast approximatation algorithm for the multiple label energy minimization problem using a cycle of α-expansion moves. It starts with an arbitrary labeling and performs iterative optimization cycles until the process converges. Each cycle consists of iterating over the set of labels, running the α-expansion move once for every label α. This involves finding a new labeling f' obtained by increasing the number of α labels, which is better than the current labeling f, ie. $E(f') \leq E(f)$. The algorithm will converge when in a particular cycle, no better f' cannot be found.

Boykov et. al. [2] analyze the algorithm and prove bounds as well as state necessary properties of the penalty functions under which the bounds are correct. In this paper, we shall only describe the max-flow graph construction for the α-expansion step and see how executing max-flow on it yields f' from f.



edge	weight	for
$t_p^{\bar\alpha}$	∞	$p \in \mathcal{P}_\alpha$
$t_p^{\bar\alpha}$	$D_p(f_p)$	$p \notin \mathcal{P}_\alpha$
t_p^{α}	$D_p(\alpha)$	$p \in \mathcal{P}$
$e_{\{p,a\}}$	$V(f_p, \alpha)$	$\{p,q\} \in \mathcal{N},\ f_p \neq f_q$
$e_{\{a,q\}}$	$V(\alpha, f_q)$	
$t_a^{\bar\alpha}$	$V(f_p, f_q)$	
$e_{\{p,q\}}$	$V(f_p, \alpha)$	$\{p,q\} \in \mathcal{N},\ f_p = f_q$

Figure 6: Graph Construction for the α-expansion step. (taken from Boykov et. al. [1]) and edge cost for the various edges in the graph are shown in a table. Note that the notation $D_p(label)$ denotes the data penalty term whereas $V(l_p, l_q)$ denotes the second order penalty term for a pixel pair $(p, q) \epsilon N$.

The graph generated for each α-expansion step, G_α will be different and will depend on the partition P of the pixels induced by the current labeling f. The vertices of this graph consist of pixels in the image and two terminal vertices denoted by α and $\bar\alpha$. For every pair of pixels p,q in the neighbourhood system N, which are labeled differently under f (ie. $f_p \neq f_q$), an *auxilliary vertex* $a_{(p,q)}$ is created with a triplet of edges to vertices p , q and $\bar\alpha$. Following

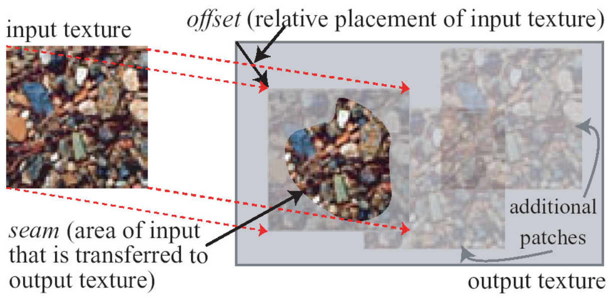

input texture
offset (relative placement of input texture)
additional patches
seam (area of input that is transferred to output texture)
output texture

Figure 7: Image Texture Synthesis example from Kwatra et. al. [4]. The texture synthesis problem involves two parts (1) Placing small patches of the input texture at various relative offsets with respect to each other (2) Computation of a seam that allows a new patch to be smoothly and 'seamlessly' merged into the existing patches.

the notation used in Section 3.2, *t-links* and *n-links* are constructed as shown in Figure 6 (1D version for simplicity). The table shows how edge capacities are set up.

A cut on this graph must sever exactly one t_p^α*-link* for every pixel p. This gives rise to a new labeling f' after all the edges belonging to the cut have been removed. Then p is labeled α if its t_p^α*-link* is intact and its old label if the t_p^α*-link* is severed.

4. 2D and 3D Texture Synthesis

4.1. Problem Statement and Goal

2D texture synthesis addresses the problem of generating larger textures from small texture samples with identical perceptual properties. Texture synthesis can also be done in 3D by treating spatio temporal texture volumes in the same way as 2D image patches. The texture synthesis problem has two parts. First multiple texture patches need to be placed relative to each other in an optimal way. Next, these patches must be combined by computing optimal seams between two neighbouring patches. Kwatra et. al. [4] compute these seams using a graph-cut approach.

4.2. Patch Fitting and Seam Computation

Figure 8 shows the graph construction for various scenarios of patch fitting. Labels in this problem indicate whether a pixel in the output texture patch needs to come from the old patch or the new texture patch. The boundary of the new patch with the old texture is the new seam that is computed. The simplest scenario is depicted in Figure 8(a) where there are only two overlapping blocks. A grid graph where vertices represent pixels is constructed for only the overlapping region in the two texture patches. There are two additional nodes A and B representing the old and the new

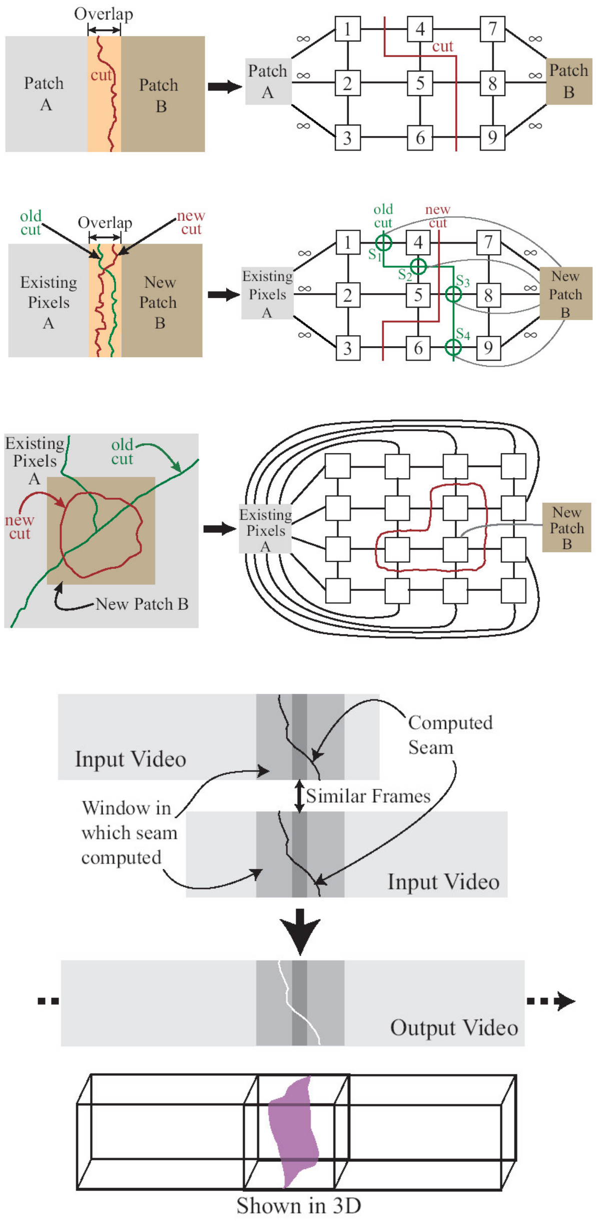


Figure 8: Graph Construction for Texture Synthesis (figures taken from Kwatra et. al. [4]). (a) Recovering the visually optimal seam by computing the minimum cut in the constructed graph (b) Computing the new seam when old seams are present in the existing patch. (2) Seamlessly merging a new patch overlaying pixels from an old texture patch. Bottom : Video texture patches are spatio temporal volumes and seams are 2D surfaces embedded in the 3D volume. Seam computation using graph-cuts is similar to the 2D case.

patch. Some of the boudary pixels of the two patches are constrained by assigning infinitely high edge weights to the edges connecting them to A and B respectively. The edge connecting pixels (p, q) in the grid graph are given the the edge cost

$$M(p, q, A, B) = ||A(p) - B(p)|| + ||A(q) - B(q)||$$

. The original texture synthesis problem is transformed into computing a minimum cost A-B cut in this graph that is equivalent to computing the most likely binary labeling that yields an imperceptible seam. The min-cut is computed by running max-flow algorithms as we have studied earlier.

The simple formulation described so far does not handle the case when old seams are present in the old texture patch. This situation illustrated in Figure 8(b) will occur when texture patches are being added iteratively to build large textures. The graph construction for this scenario is slightly different. The node B is still used to represent the new patch whereas A represents the collection of old patches present in the current texture. This graph contains additional nodes called *seam nodes* (see Figure 8(b)) which correspond to the old seams, computed during previous graph cut iterations. Each seam node s_{pq} between pixels p and q is triply connected to B, p and q through edges with weights $M(p, q, A_p, A_q)$, $M(p, q, A_p, B)$ and $M(p, q, B, A_q)$ respectively. A min-cut on this new graph will automatically decide whether to leave the old seam intact (edges between such seam nodes and B will be intact after the min-cut edges have been severed) or replace it by the new seam at the same pixels if either of the edges between s_{pq} and its neighbours p and q are severed.

The formulation for video textures (see Figure 8) is identical in all respects except that the texture patches are now volumes and hence the grid graph is a 3D grid graph with a 6-connected neighbourhood system.

The third scenario which is applicable to novel image synthesis shown in Figure 8(c) involves overlaying a new texture patch over an existing patch and seamlessly merging it into the old texture. Once again the graph constructed corresponds to the pixels in the overlapping region and the edges to the terminal nodes A and B which are left intact after the min-cut partitions the graph decide which patch contributes information for that particular pixel. Seams are usually more noticeable in low frequency regions, and better seams can be computed by modifying the edge weighting function to take image gradients into account. In such a case $M(p, q, A, B)$ is scaled by the magnitude of the one of the component of the gradient depending on whether the concerned edge is aligned horizontally, vertically or temporally (for video textures). This penalizes the cut from passing through a low frequency region in the image or video.

5. Learning from Labeled and Unlabeled Data

5.1. Problem Statement and Goal

In the domain of machine learning and pattern classification, labeled training data is often at a premium, whereas unlabeled examples are available in plenty for most practical problems. If this unlabeled data could be automatically labeled based on the smaller labeled training set, a better classifier could be built, one which uses all the training data. The primary challenge here is to achieve accurate labeling even when the set of labeled examples is small. The approach towards solving this problem involves building a graph on the whole training set using edge weights to represent similarity between examples and then partitioning the graph in a way that best explains the known labeled examples. Blum et. al [5, 6] perform this partitioning using randomized minimum-cuts on the constructed graphs and provide theoritical justification for their chosen approach.

5.2. Graph Mincuts Approach

Given a set of (positive and negative) labeled examples L, and a set of unlabeled examples U in a training set, the goal is to find a labeling of the examples in U that minimizes the leave-one-out cross validation (LOOCV) error of a nearest neighbour learning algorithm, when applied to the dataset $L \bigcup U$. The set L, will consist of mutually exclusive subsets L_+ and L_-, the sets of posivite and negative labeled examples respectively. The graph-cut approach could be used to label examples in U, if a graph could be constructed where edge weights are assigned using the notion of 'pairwise distance', (ie. similar examples are connected through edges with large weights whereas dissimilar example pairs have edges with small weights). The minimum cost cut would then partition or cluster the positively labeled examples from the negatively labeled ones in a nearest-neighbour sense.

The basic mincut algorithm works as follows. A graph $G(V, E)$ is constructed where $V = L \bigcup U \bigcup \{v_+, v_-\}$and $E \subset VXV$. Next edges between labeled vertices and the terminal vertices (v_+, v_-) are set up as follows : $w(v, v_+) = \inf \ \forall \ v \ \epsilon \ L_+$ and $w(v, v_-) = \inf \ \forall \ v \ \epsilon \ L_-$.

Edges between vertices representing all examples (both labeled and unlabeled) are assigned some weight based on a suitable edge weighting function. A minimum cost (v_+, v_-) cut is then computed on G using a max-flow algorithm by treating v_+ as the source and v_- as the sink and all the edge weights as capacities. If there are multiple min-cuts, the algorithm chooses one of them using some well defined criteria. By removing the edges constituting the min-cut, V is partitioned into V_+ and V_-, the set of positive and negatively labeled examples respectively.

Blum et. al. proved that this graph min-cut approach would produce a labeling with a bounded error. This error would be the minimum LOOCV error for an *averaging* k-nearest neighbour algorithm (one that predicts a label for a new example, based on the weighted average of the labeled examples, for instance distance to the labeled examples). LOOCV criteria is often used to evaluate the performance of a classification algorithm and is computed as follows. A single example is chosen to represent the test data, while all the other examples make up the training set for the particular classification algorithm. Based on this training set, the single example is classified and the success of classification is recorded. When the above process is repeated for every example in the original dataset, the total number of mis-classifications is the LOOCV error.

The construction of G proceeds as follows, some metric (Euclidean distance L_2, Hamming distance L_0 etc.) are used to compute pair-wise distances. If two examples are closer than a distance parameter δ, they are connected by an edge with the corresponding distance as the edge weight. Various strategies for choosing δ are discussed in [6, 5] (for instance choosing δ such that size of the largest component in G was larger than half the size of $L \bigcup U$ worked well in practice).

5.3. Randomized Mincuts

This graph cut framework has some limitations as far as labeling accuracy is concerned. If multiple min-cuts exist in the constructed graph, the choice for the best min-cut is unclear and the consequent partitioning could be unbalanced (few unlabeled examples are assigned to V_+ and many to V_- or vice versa). This is likely to happen when the labeled set is small. Also if the graph had a large number of disconnected components, the examples in those components could be labeled ambiguously.

The Randomized Min-cuts algorithm [5] tries to address these weaknesses. Artificial random noise is added to the edge weights in the graph (described in Section 5.2) and min-cuts are repeatedly computed. The graph topology remains the same for the multiple min-cut executions. The algorithm votes amongst all the min-cuts and ends up choosing the one in the middle, considering how many times a particular example was labeled positive and how many time it was labeled negative. This gives rise to better accuracy coverage statistics (allows one to set a confidence based on how many min-cuts agree on the labeling of a particular example). Morever highly unbalanced partitions are discarded.

Randomization has been shown to be useful [5] only when small well balanced cuts (boundaries) exist in the dataset. Thus the graph construction approach needs to be conservative by assigning edges between examples only when they are very similar. Two approaches which have worked well in practice are (1) building a minimum spanning tree (MST) on the whole dataset using pair-wise distance as edge weights (this yields a connected, sparse graph). (2) building a δ-MST, where examples ares connected if they are within a distance δ of each other. Extra edges are added to the graph by treating the components as nodes and computing their MST.

6. Motivation for using Graph-cuts

All the problems that have been solved using graph-cuts have one property in common, they are all optimizing labelings ie. trying to compute the most likely labeling from a very large solution space to explain some measured data. These labels are either intensity values, disparity values in images, objects in images, or distinct classes in large training datasets. The individual entities which are labeled are represented as sites in the graphs and the contextual information whch constrains the labelings are used to setup edges in the graph. In other words the probability of a particular site getting a specific label depends on the labelings of its neighbours. This contextual information is rich in images, video, textures, large training sets and is exploited in the graph cuts framework. The notion of a cut in such a graph is analogous to the idea of partitioning the sites using labels. The graph construction needs to be done in a way such that the minimum cut will yield the most likely labeling or partitioning of the measured data. The use of graph-cuts in the various papers studied [1, 4, 5] were justified from a MRF perspective.

While applying graph-cuts to a new problem, the assumptions made in the problem transformation could place restrictions on the solutions produced. These would in general be problem dependent and needs to be well understood. For eg. Boykov [2] realises that graph-cuts can only deal with only a certain class of energy functions in the MRF framework and Blum et. al. [5] theoritically justify that the min-cut approach would have the same classification error as a class of nearest neighbour learning algorithms.

7. Comparisons

The application of graph-cuts to texture synthesis, [4] and semi-supervised learning [5] were motivated by the success of the energy minimization framework [1] in vision. Roy et. al. [3] had earlier proposed a solution to the n-view 3D reconstruction problem by directly formulating it as a max-flow min-cut problem instead of converting it first into an energy minimization problem. Although they construct a similar graph compared to Boykov et. al. [1], their edge weighting function does not guarantee optimality properties.

In problems encountered in vision and graphics, there is often an underlying regular grid structure with a clearly defined notion of neighbourhood in this grid itself. This seems to set the basic topology of the graph on which graph-cuts are applied. In contrast, interesting issues arise in the semi-supervised learning problem, as far as graph construction is concerned. The topology of the graph and the edge weighting function are harder to choose and require an in-depth analysis of properties of nearest neighbour classification algorithms. Constructing a minimum spanning tree on the whole training set for the randomized min-cut algorithm was proven to be a good choice although less intuitive compared to the other graph-cut formulations.

In some sense, the application of graph cuts to texture synthesis is dual to its applications to vision problems. Graph-cuts find optimal seams in texture synthesis by trying to find similar pixels from different patches (the cut goes between them). However the min-cuts in vision typically coincide with some form of visual boundaries, either intensity edges, disparity edges (depth-discontinuities) or segmentation boundaries. The choice of the edge weighting function in the graph is what defines the behaviour of the cut from problem to problem.

Similarly the formulation of the image synthesis scenario [4], Figure 8(c) as a graph-cut problem is inverse to the the object segmentation problem in images described in Section 3.5. Note that Kwatra's technique for integrating the new patch into a texture which already contains multiple old patches and seam is similar to a single α-expansion step in Boykov's algorithm (Section 3.6).

The Max-flow problem which is a dual of the mincut problem, has no direct link with the multiway cut problem where there is no clear notion of flow. However interestingly, the α-expansion algorithm (see Section 3.6), an approximation algorithm for the multiway cut problem uses multiple iterations of max-flow. This is why max-flow algorithms are key to solving graph-cut problems.

7.1. A New Max-flow / Min-cut algorithm

It was observed that the graphs encountered within the vision and graphics problem domain, were typically sparsely connected regular grid graphs with connections between grid neighbours and often contained a large number of connections to the source and the sink. Although efficient max-flow algorithms have been developed in the past to deal with special graphs (planar graphs etc.), no efficient algorithms existed for these kind of graphs. Boykov et. al. [1] studied the performance of standard max-flow algorithms on the 'vision' graphs and realised that the augmenting path algorithms could be modified to perform better on them. Most augmenting path style algorithms always recompute the shortest path from scratch after an augmenting path is computed in the residual graph (see Section: 2.2. This is a costly operation on large grid graphs since the breadth first search needs to visit all vertices. They decided to reuse search trees in order to compute source-sink paths for multiple iterations instead of building it from scratch everytime.

This was done by maintaining two different search trees, one rooted at the source, and one at the sink. By growing the trees, an augmenting path from source to sink is found when the two trees touch each other. After the flow is augmented along this path, the edges constituting the path are removed and the two search trees get partitioned into forests. Subsequently the tree structures are restored through a series of operations which consist of either finding a parent for an orphaned node (whose parent link was removed as part of the augmenting path) or by freeing the orphaned node (such nodes are added back to the trees, during the growth phase of the algorithm.

Although this algorithm works much faster in practice, it does not have a strong upper bound, its worse case time is $O(n^2m|C|)$ where $|C|$ is the maximum capacity, however on most sparsely connected grid graphs, it beats the best know push-relabel style algorithm by a factor of 2-5.

8. Summary and Conclusions

The key contribution of each of the papers studied here [2, 4, 5] is an innovative reformulation of the original problem into a well understood graph cut and network flow problem for which well-known algorithms are known to exist. Using clever transformations, the solution to the original problems obtained using the graph cut approach was shown to be optimal under valid assumptions. Motivated by the application of graph-cuts and max-flow to computer vision, a new max-flow/min-cut algorithm was proposed, which was shown to be faster than the best known max-flow algorithms for the special grid graphs which are encountered in the domain of Vision and Graphics. This integrative paper studied the basic theory of the max-flow problem and the classes of algorithms used to solve it. It then presented the transformations of each of these problems into the graph-cut problems which were thereby solved using max-flow algorithms. Finally, the common thread between all the problems were investigated and the various graph-cut formulations were compared against one other. This was done in order to provide insight into the power of the technique, so that it could be applied to more diverse problems in future.

References

[1] Y. Boykov and V. Kolmogorov, "An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision", *In IEEE transactions on Pattern Analysis and Machine Intelligence (PAMI), vol 26, no.9, pp 1124-1137, Sept 2004.*

[2] Y. Boykov, O. Veksler and R. Zabih, "Faster approximate energy minimization via graph cuts", *In IEEE transactions on*

Pattern Analysis and Machine Intelligence (PAMI), vol 23, no. 11, pp 1222-1239, Nov 2001.

[3] S. Roy and I.J. Cox, “A maximum-flow formulation of the n-camera stereo correspondence problem”, *ICCV 98, pp. 492-502, 1998.*

[4] V. Kwatra, A. Schodl, I. Essa, G. Turk and A. Bobick, “Graphcut Textures: Image and Video Synthesis Using Graph Cuts”, *In SIGGRAPH 2003, pp. 277-286.*

[5] A. Blum, J. Lafferty, M.R. Rwebangira and R. Reddy, “Semi-Supervised Learning Using Randomized Mincuts”, *In Proceedings of the 21st International Conference on Machine Learning, Banff, Canada 2004.*

[6] A. Blum and S. Chawla, “Learning from labeled and unlabeled data using graph mincuts”, *In Proceedings of the 18th International Conference on Machine Learning, 2001.*

[7] A.V Goldberg and R. E. Tarjan, “A new approach to the maximum-flow problem”, *Journal of the Association for Computing Machinery, vol 35, no. 4, pp 921-940, Oct 1988.*

[8] L. Ford and D. Fulkerson, “Flow in Networks”, *Princeton University Press, 1962.*

[9] “Algorithm for solution of a problem of maximum flow in networks with power estimation”, *Soviet Math. Dokl., vol 11, pp 1277-1280, 1970.*

[10] T.H. Cormen, C.E. Leiserson and R.L. Rivest, “Introduction to Algorithms”, *McGraw-Hill, 1990.*

[11] S. Z. Li, “Markov Random Field Modeling in Computer Vision”, *Springer Verlag, 1995.*