Dynamic Graph Cuts for Efficient Inference in Markov Random Fields

Pushmeet Kohli, *Member, IEEE* and Philip H. S. Torr, *Senior Member, IEEE*

***Abstract*— In this paper we present a fast new fully dynamic algorithm for the st-mincut/max-flow problem. We show how this algorithm can be used to efficiently compute MAP solutions for certain dynamically changing MRF models in computer vision such as image segmentation. Specifically, given the solution of the max-flow problem on a graph, the dynamic algorithm efficiently computes the maximum flow in a modified version of the graph. The time taken by it is roughly proportional to the total amount of change in the edge weights of the graph. Our experiments show that, when the number of changes in the graph is small, the dynamic algorithm is significantly faster than the best known static graph cut algorithm. We test the performance of our algorithm on one particular problem: the object-background segmentation problem for video. It should be noted that the application of our algorithm is not limited to the above problem, the algorithm is generic and can be used to yield similar improvements in many other cases that involve dynamic change.**

***Index Terms*— Energy Minimization, Markov Random Fields, Dynamic graph cuts, Maximum flow, st-mincut, Video segmentation.**

I. Introduction

Graph cuts have been extensively used in computer vision to compute the maximum a posteriori (MAP) solutions for various discrete pixel labelling problems such as image restoration, segmentation, voxel occupancy and stereo [17], [18], [24], [26]–[30], [36], [38]. One of the primary reasons behind their growing popularity is the availability of efficient algorithms with low polynomial time algorithmic complexity for computing the maximum flow (max-flow) in graphs of arbitrary topology [1], [5]. These algorithms enable fast computation of the minimum cost st-cut (st-mincut) problem, which in turn allows for the computation of globally optimal solutions for important classes of energy functions [12], [14], [16], [23], [25]. This includes sub-modular functions of binary random variables which have been successfully used for formulating a wide range of problems [4], [32].

Greig *et al.* [14] were one of the first to use graph cuts in computer vision. They showed that if the pairwise potentials of a two label *pairwise* Markov Random Field (MRF) were defined as an Ising model, then the exact maximum a-posteriori (MAP) solution can be obtained in polynomial time by solving a st-mincut problem. The use of Graph cuts has since been extended to MRFs with multiple labels (see [6], [16] and [30]). This equivalence between st-mincut and MAP-MRF estimation makes graph cuts extremely important.

Pushmeet Kohli is at the Department of Computing, Oxford Brookes University, Oxford OX33 1HX, UK. E-mail: pushmeet.kohli@brookes.ac.uk.

Philip H. S. Torr is with the Department of Computing, Oxford Brookes University, Oxford OX33 1HX, UK. E-mail: philiptorr@brookes.ac.uk. Webpage: http://cms.brookes.ac.uk/staff/PhilipTorr/

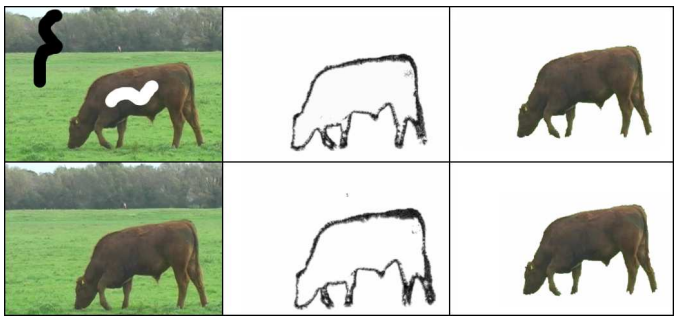

Fig. 1. *Dynamic image segmentation using Graph Cuts. The images in the first and third column are two consecutive frames of a video sequence and their respective segmentations. The first image in the first column also shows the user segmentation seeds (pixels marked by black (background) and white (foreground) colours). The user marked image pixels are used to learn histograms modelling foreground and background likelihoods (as in [4]). In column 2, we observe the n-edge flows obtained corresponding to the* MAP *solutions of the* MRFs *used for formulating the image segmentation problem on the two frames. It can be clearly seen that the flows corresponding to the two segmentations are similar. The flows from the first segmentation were used as an initialization for the max-flow problem corresponding to the second frame. The time taken for this procedure was much less than that taken for finding the flows from scratch.*

In many real world applications, multiple similar instances of a problem need to be solved sequentially e.g. performing image segmentation on the frames of a video. The data (image) in this problem changes from one time instance to the next. Given the solution to an instance of the problem, the question arises as to whether this solution can help in solving other similar instances. In this paper we answer this particular question for energies that can be solved exactly using graph cuts. Specifically, we show how the solution to the max-flow problem corresponding to an MRF can be used in solving another *similar* MRF with slightly different energy terms.

Our algorithm records the flow obtained during the computation of the max-flow corresponding to a particular problem instance. This recorded flow is used as an initialization in the process of finding the max-flow solution corresponding to the new problem instance (as seen in figure 1). Our algorithm belongs to a broad category of algorithms which are referred to as *dynamic*. These algorithms solve a problem by dynamically updating the solution of the previous problem instance. Their goal is to be more efficient than a recomputation of the solution after every change from scratch. Given a directed weighted graph, a *fully* dynamic algorithm should allow for unrestricted modification of the graph. This involves addition and deletion

of nodes and edges in the graph as well as modification of the cost (capacity) of any graph edge.

Overview of Dynamic Graph Cuts: Dynamic algorithms are not new to computer vision. They have been extensively used in computational geometry for problems such as range searching, intersections, point location, convex hull, proximity and many others. For more on dynamic algorithms used in computational geometry, the reader is referred to [8]. A number of algorithms have been proposed for the dynamic mincut problem. Thorup [33] proposed a method which had a $O(\sqrt{m})$ update time and took $O(\log n)$ time per edge to list the cut edges where n and m are the number of nodes and edges in the graph respectively. However, the dynamic *st-mincut* problem has remained relatively ignored.

Gallo *et al.* [13] introduced the problem of parametric max-flow and used a partially dynamic graph cut algorithm for the problem. Their algorithm had a low polynomial time complexity but was unable to handle arbitrary changes in the graph. Recently, Cohen and Tamassia [9] proposed a dynamic algorithm for the problem by showing how dynamic expression trees can be used for maintaining st-mincuts with $O(\log m)$ time for update operations. However, their algorithm could only handle series-parallel diagraphs[1].

Boykov and Jolly [4] were the first to use a *partially* dynamic st-mincut algorithm in a vision application by proposing a technique with which they could update capacities of t-edges of *certain* graph edges, and recompute the st-mincut dynamically. They used this method for performing interactive image segmentation, where the user could improve segmentation results by giving additional segmentation cues (seeds) in an online fashion. Specifically, they described a method for updating the cost of t-edges in the graph.

In this paper we present a new fully dynamic algorithm for the st-mincut problem which allows for arbitrary changes in the graph[2]. We show how this algorithm can be used to *dynamically* perform MAP inference in an MRF. Such an inference procedure is extremely fast and has been used for a number of problems [7], [15], [21]. Recently, Juan and Boykov [19] proposed an algorithm in which instead of *reusing flow*, they reused the st-mincut solution corresponding to the previous MRF instance to generate an initialization.

Organization of the Paper: An outline of the paper follows. Section II provides an overview of the st-mincut/maxflow problem. In section III we show how MRFs can be used to formulate labelling problems such as image segmentation. The procedure for minimizing energy functions using graph cuts and the relationship between energy and graph reparameterization is explained in section IV. Section V shows how MAP solutions of dynamically changing MRFs can be efficiently computed by reusing flow. Specifically, it describes how the residual graph can be transformed to reflect the changes in the original graph using graph reparameterization, and discusses issues related to the computational complexity of the algorithm. In section VI, we describe how the process of recomputing the st-mincut/max-flow can be further optimized by using *recycled* search trees. The procedure for performing image segmentation on video sequences is described in section VII-A. In the same section we compare the performance of our dynamic algorithm with that of the st-mincut algorithm described in [5].

[1]Series-Parallel digraphs are graphs which are planar, acyclic and connected.

[2]The first version of this paper appeared as [20].

II. Preliminaries

In this section we provide a general overview of the st-mincut/maxflow problem, and give the notation used in the paper. A directed weighted graph $G(V, E, C)$ with non-negative edge weights, is defined by a set of nodes V, a set of directed edges E, and an edge cost function $C : E \rightarrow \mathbb{R}$ which maps each edge (i, j) to a real number c_{ij}[3]. We will use n and m denote the number of nodes $|V|$ and the number of edges $|E|$ in the graph respectively. Graphs used in the st-mincut problem have certain special nodes called the terminal nodes, namely the source s, and the sink t. The edges in the graph can be divided into two disjoint categories: t-edges which connect a node to a terminal node, and n-edges which connect nodes other than the terminal nodes with each other. We make the following assumptions in our notation: $(i, j) \in E \Rightarrow (j, i) \in E$, and $(s, i) \in E \ \wedge \ (i, t) \in E$ for all $i \in V$. These assumptions are non-restrictive as edges with zero edge weights are allowed in our formulation. Thus we can conform to our notation without changing the problem.

A *cut* is a partition of the node set V into two parts S and $\overline{S} = V - S$, and is defined by the set of edges (i, j) such that $i \in S$ and $j \in \overline{S}$. The cost of the cut $(S, \overline{S})$ is given as: $C_{S,\overline{S}} = \sum_{i \in S, j \in \overline{S}} c_{ij}$. An st-cut is a cut satisfying the properties $s \in S$ and $t \in \overline{S}$. Given a directed weighted graph G, the st-mincut problem is that of finding a st-cut with the smallest cost. By the Ford-Fulkerson theorem [11], this is equivalent to computing the maximum flow from the source to the sink with the capacity of each edge equal to c_{ij} [1].

Formulating The Max-Flow Problem: For a network $G(V, E)$ with a non-negative capacity c_{ij} associated with each edge, the max-flow problem is to find the maximum flow f from the source node s to the sink node t subject to the edge capacity and mass balance constraints:

$$0 \leq f_{ij} \leq c_{ij} \quad \forall (i, j) \in E, \quad \text{and} \tag{1}$$

$$\sum_{i \in \ N(x)} (f_{xi} - f_{ix}) = 0 \quad \forall x \in V \backslash \{s, t\} \tag{2}$$

where f_{ij} is the flow from node i to node j and $N(x)$ is the neighbourhood of x i.e. $N(x)$ consists of all nodes connected by an edge to x [1].

Observe that we can initialize the flows in the t-edges of any node x of the graph as $f_{sx} = f_{xt} = \min(c_{sx}, c_{xt})$. This corresponds to pushing flow through these edges from the source to the sink and has no effect on the final solution of the st-mincut problem. From this it can be deduced that the solution of the st-mincut problem is invariant to the absolute value of the terminal edge capacities c_{sx} and c_{tx}. It only depends on the difference of these capacities $(c_{xt} - c_{sx})$.

[3]In the paper we restrict our attention to edge cost functions of the form $C : E \rightarrow \mathbb{R}^{+} \cup \{0\}$.

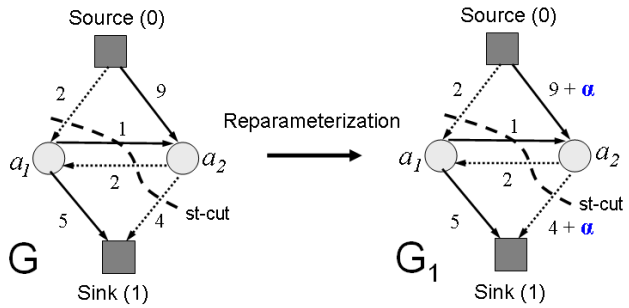

Fig. 2. *Graph Reparameterization. The figure shows a graph G its reparameterization G_1 obtained by adding a constant α to both the t-edges of node a_2. Observe that although the cost of the st-mincut in G and G_1 is different, the st-mincut includes the same edges for both graphs and thus induces the exact same partitioning of the graph.*

Adding or subtracting a constant to these capacities changes the objective function by a constant and does not effect the overall st-mincut solution as can be seen in figure 2. Such transformations result in a reparameterization of the graph and will be explained later in the paper.

Augmenting Paths, Residual Graphs: Given a flow f_{ij}, the residual capacity r_{ij} of an edge $(i,j) \in E$ is the maximum additional flow that can be sent from node i to node j using the edges (i,j) and (j,i) or formally $r_{ij} = c_{ij} - f_{ij} + f_{ji}$. A residual graph $G(f)$ of a weighted graph G consists of the node set V and the edges with positive residual capacity (with respect to the flow f). An augmenting path is a path from the source to the sink along unsaturated edges of the residual graph.

III. Markov Random Fields

MRFs for labelling problems are described next followed by some examples in vision where dynamic MRFs occur. Consider a random field consisting of a set of discrete random variables $\{x_1, \ldots, x_n\}$ defined on the set $\mathcal{V}$, such that each variable x_v takes values from the label set $\mathcal{X}_v$. The set of all variables $x_v, \forall v \in \mathcal{V}$ is represented by the vector $\mathbf{x}$ which takes values from the set $\mathcal{X}$ defined as $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \ldots \times \mathcal{X}_n$. $\mathcal{N}_v$ will be used to denote the set consisting of indices of all variables which are neighbours of the random variable x_v in the graphical model. If each configuration $\mathbf{x}$ is assigned a probability $\Pr(\mathbf{x})$, then the random field defined above is said to be a MRF [37] with respect to a neighborhood $\mathcal{N} = \{\mathcal{N}_v | v \in \mathcal{V}\}$ if and only if it satisfies the positivity property $\Pr(\mathbf{x}) > 0 \quad \forall \mathbf{x} \in \mathcal{X}$, and the Markovian property:

$$\Pr(x_v | \{x_u : u \in \mathcal{V} - \{v\}\}) = \Pr(x_v | \{x_u : u \in \mathcal{N}_v\}) \ \ \forall v \in \mathcal{V}. \tag{3}$$

We follow the notation of [22] and formulate the MAP-MRF estimation problem as an energy minimization problem. The energy corresponding to a MRF configuration $\mathbf{x}$ is defined as the negative log likelihood of its joint posterior probability as:

$$E(\mathbf{x}|\theta) = -\log \Pr(\mathbf{x}|\mathbf{D}) - \text{const.} \tag{4}$$

Here θ is the energy parameter vector defining the MRF energy and is derived from the data. The energy of a configuration for such a pairwise MRF can be written in terms of unary and pairwise energy terms as:

$$E(\mathbf{x}|\theta) = \sum_{v \in \mathcal{V}} \left(\phi(x_v) + \sum_{u \in \mathcal{N}_v} \phi(x_u, x_v) \right) + \text{const.} \tag{5}$$

$\psi(\theta)$ will be used to denote the value of the energy of the MAP configuration of the MRF and is defined as:

$$\psi(\theta) = \min_{\mathbf{x} \in \mathcal{X}} E(\mathbf{x}|\theta). \tag{6}$$

A. MRFs for Image Segmentation

In the context of image segmentation, $\mathcal{V}$ corresponds to the set of all image pixels, $\mathcal{N}$ is a neighbourhood defined on this set[4], each set $\mathcal{X}_v$ comprises of the labels $\{l_1, \ldots, l_L\}$ representing the different image segments, and the random variables in the set $\mathbf{x}$ denote the labelling of the pixels in the image. Note that every possible assignment of the random variables $\mathbf{x}$ (or configuration of the MRF) defines a segmentation. The image segmentation problem can thus be solved by finding the least energy configuration of the MRF. The energy corresponding to a configuration $\mathbf{x}$ consists of a likelihood and a prior term as:

$$\Psi_1(\mathbf{x}) = \sum_{v \in \mathcal{V}} \left(\phi(\mathbf{D}|x_v) + \sum_{u \in \mathcal{N}_v} \psi(x_u, x_v) \right) + \text{const}, \tag{7}$$

where $\phi(\mathbf{D}|x_v)$ is the log likelihood which imposes individual penalties for assigning label l_k to pixel v and is given by

$$\phi(\mathbf{D}|x_v) = -\log \Pr(v \in \mathcal{S}_k | \mathcal{H}_k) \qquad \text{if } x_v = l_k. \tag{8}$$

Here $\mathcal{H}_k$ is the RGB distribution for $\mathcal{S}_k$, the segment denoted by label l_k, $\Pr(v \in \mathcal{S}_k | \mathcal{H}_k) = \Pr(I_v | \mathcal{H}_k)$ and I_v is the colour of the pixel v. The prior $\psi(x_u, x_v)$ takes the form of a Generalized Potts model:

$$\psi(x_u, x_v) = \begin{cases} K_{uv} & \text{if} \quad x_u \neq x_v, \\ 0 & \text{if} \quad x_u = x_v. \end{cases} \tag{9}$$

In MRFs used for image segmentation a contrast term is added which favours pixels with similar colour having the same label [2] [4] [6] [17]. This is incorporated in the energy function by reducing the cost within the Potts model for two labels being different in proportion to the difference in intensities of their corresponding pixels. For instance, for the experiments discussed in section VII-A, we use the term

$$\gamma(u,v) = \lambda \exp\left(\frac{-g^2(u,v)}{2\sigma^2} \right) \frac{1}{\text{dist}(u,v)}, \tag{10}$$

where $g^2(u,v)$ measures the difference in the RGB values of pixels u and v, $\text{dist}(u,v)$ gives the spatial distance between u and v and σ is a model parameter. This is a likelihood term (not prior) as it is based on the data. The energy function of the MRF now becomes:

$$\Psi_2(\mathbf{x}) = \sum_{v \in \mathcal{V}} \left(\phi(\mathbf{D}|\mathbf{x}_v) + \sum_{v \in N_u} (\phi(\mathbf{D}|\mathbf{x}_u, \mathbf{x}_x) + \psi(\mathbf{x}_u, \mathbf{x}_v)) \right). \tag{11}$$

[4]For our experiments, we have used the standard 8-neighbourhood.

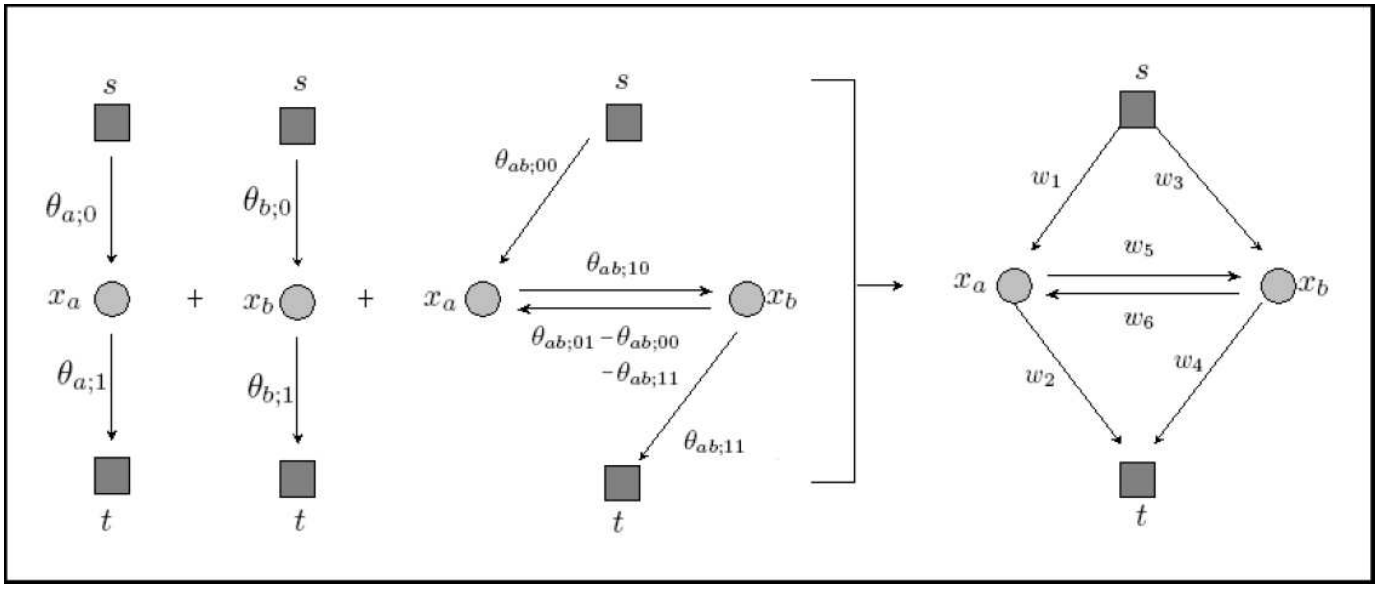

Fig. 3. *Energy minimization using graph cuts. The figure shows how individual unary and pairwise terms of an energy function taking two binary variables are represented and combined in the graph. Multiple edges between the same nodes are merged into a single edge by adding their weights. For instance, the cost w_1 of the edge (s, x_a) in the final graph is equal to: $w_1 = \theta_{a;0} + \theta_{ab;00}$. The cost of a st-cut in the final graph is equal to the energy $E(\mathbf{x})$ of the configuration $\mathbf{x}$ the cut induces. The minimum cost st-cut induces the least energy configuration $\mathbf{x}$ for the energy function.*

The contrast term of the energy function is defined as

$$\phi(\mathbf{D}|\mathbf{x}_u, \mathbf{x}_v) = \begin{cases} \gamma(u,v) & \text{if} \quad x_u \neq x_v \\ 0 & \text{if} \quad x_u = x_v. \end{cases} \tag{12}$$

B. Dynamic Markov Random Fields

It can be observed that the energy Ψ_2 of the MRF defined above for the image segmentation problem is dependent on the data $\mathbf{D}$ (the colour of the pixels of the image to be segmented) and the parameters used in the energy function[5]. This dependence results in any change in the data causing a change in the energy function of the MRF. In many vision problems the data infact does change with time resulting in changes in the energy function of the MRF. We refer to MRFs used for formulating such problems as being *dynamic*. For instance, this is the case when image segmentation is performed on the frames of a video.

As the image (data) changes slightly from one frame to the next it might be hoped that the solution of the problem at the first frame can be used to speed up computation at the second. We show that this conjecture is infact correct. The key contribution of this paper is the dynamic max-flow algorithm with which a solution of a dynamic MRF can be efficiently computed using the solution of its previous state (see figure 1).

IV. Energy Minimization and Graph Reparameterization

The most probable or the MAP configuration of a MRF is the configuration having the least energy. Certain classes of energy functions can be minimized exactly using graph cuts. In this section we briefly describe the process of energy minimization. We then explain the concept of Graph Reparameterization which will be used later to explain how we can minimize dynamic energy functions.

A. Energy Minimization using Graph Cuts

The procedure for energy minimization using graph cuts comprises of building a graph in which each cut defines a configuration $\mathbf{x}$. The cost of a cut is equal to the energy $E(\mathbf{x}|\theta)$ of its corresponding configuration $\mathbf{x}$. Finding the minimum cost st-cut in this graph thus provides us with the the configuration having the least energy. Kolmogorov and Zabih [25] showed how and under what conditions energies like (5) with binary random variables can be minimized exactly using st-mincuts. Later, [16] defined a set of conditions under which problems with multiple labels can be solved exactly.

We now explain the graph construction for minimizing energies involving binary random variables. These function need to be sub-modular to be solved exactly using graph cuts [3], [25]. We use the notation of [22] and write the MRF energy (5) as:

$$\begin{aligned} E(\mathbf{x}|\theta) &= \theta_{\text{const}} + \sum_{v \in V, i \in \mathcal{X}_v} \theta_{v;i} \delta_i(x_v) \\ &+ \sum_{(s,t) \in E, (j,k) \in (\mathcal{X}_s, \mathcal{X}_t)} \theta_{st;jk} \delta_j(x_s) \delta_k(x_t), \end{aligned} \tag{13}$$

where $\theta_{v;i}$ is the penalty for assigning label i to latent variable x_v, $\theta_{st;ij}$ is the penalty for assigning labels i and j to the latent variables x_s and x_t and each $\delta_j(x_s)$ is an indicator function, which is defined as:

$$\delta_j(x_s) = \begin{cases} 1 & \text{if } x_s = j, \quad \text{where } j \in \mathcal{X}_s \\ 0 & \text{otherwise.} \end{cases}$$

[5]This also holds true for other problems which are formulated using MRFs such as the stereo labelling problem.

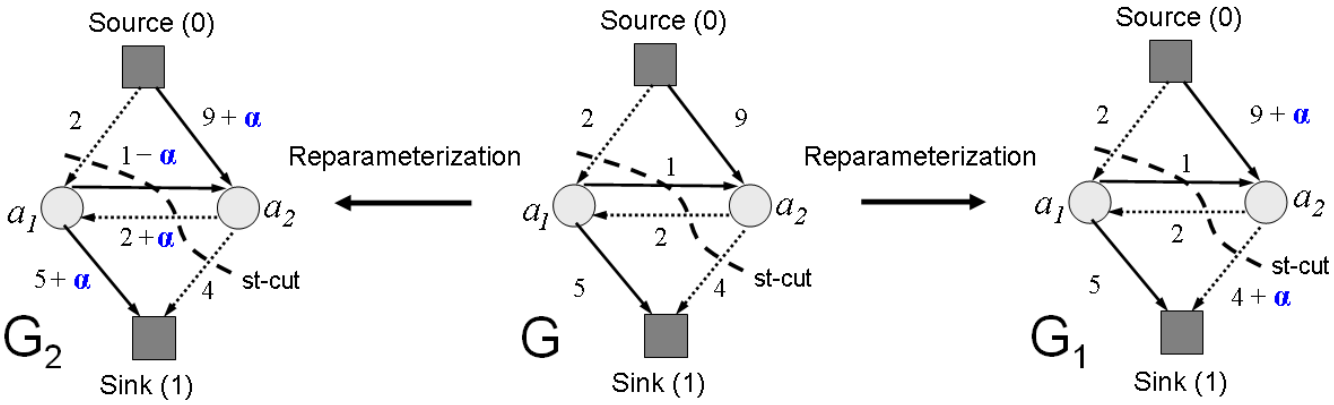

Fig. 4. *Graph Reparameterization. The figure shows a graph G, its two reparameterizations G_1 and G_2 along with their respective st-mincuts. The edges included in the st-mincut are marked by dashed lines. The reparameterized graphs G_1 and G_2 are a results of two different valid transformations of graph G. It can be clearly seen that reparameterized graphs G_1 and G_2 have the same st-mincut as graph G.*

The individual unary and pairwise terms of the energy function are represented by weighted edges in the graph. Multiple edges between the same nodes are merged into a single edge by adding their weights. The graph construction for a two node MRF is shown in figure 3. The constant term θ_{const} of the energy does not depend on $\mathbf{x}$ and thus is not considered in the energy minimization procedure. The st-mincut in this graph provides us with the MAP solution. The cost of this cut corresponds to the energy of the MAP solution. The labelling of a latent variable depends on the terminal it is disconnected from by the minimum cut[6].

B. Energy and Graph Reparameterization

Energy parameter vectors θ_1 and θ_2 are called reparameterizations of each other if and only if $\forall x, E(\mathbf{x}|\theta_1) = E(\mathbf{x}|\theta_2)$ [3], [22], [31], [34]. Note that this simply means that all possible labellings $\mathbf{x}$ have the same energy under both parameter vectors θ_1 and θ_2, and does not imply that $\theta_1 = \theta_2$. There are a number of transformations which can be applied to a energy parameter vector θ to obtain its reparameterization $\overline{\theta}$. For instance the transformations given as:

$$\forall i \quad \overline{\theta}_{v,i} = \theta_{v,i} + \alpha, \quad \overline{\theta}_{const} = \theta_{const} - \alpha \text{ and} \tag{14}$$

$$\forall i, j \quad \overline{\theta}_{st,ij} = \theta_{st,ij} + \alpha, \quad \overline{\theta}_{const} = \theta_{const} - \alpha \tag{15}$$

result in the reparameterization of the energy parameter vector.

As both parameters θ and $\overline{\theta}$ define the same energy function, the minimum energy labelling for both will be the same i.e.

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} E(\mathbf{x}|\theta_1) = \arg\min_{\mathbf{x}} E(\mathbf{x}|\theta_2) \tag{16}$$

This means that the graphs constructed for minimizing the energy functions $E(\mathbf{x}|\theta_1)$ and $E(\mathbf{x}|\theta_2)$ (using the procedure explained in the previous subsection) will have the same st-mincut. We call these graphs reparameterizations of each other. For any transformation of the energy function which results in such a *reparameterization* we can derive a corresponding transformation for a graph. Under these transformations the resulting graph will be a reparameterization of the original graph and thus will have the same st-mincut. The graph transformations corresponding to energy transformations given by equations (14) and (15) are shown in figure 4.

The transformations given above are not the only way to obtain a reparameterization. In fact pushing flow through any path in the graph can be seen as performing a valid transformation. The residual graph resulting from this flow is a reparameterization of the original graph where no flow was being passed. This can be easily observed from the fact that the residual graph has the same st-mincut as the original graph. In the next section we show how the property of graph reparameterization can be used for updating the residual graph when the original graph has been modified and the st-mincut needs to be recomputed.

V. Recomputing MAP solutions for Dynamic MRFs

This section describes one of the primary contributions of this paper. Having shown how energy functions defining certain pairwise MRFs can be minimized exactly by solving a st-mincut/maxflow problem, we now show how this max-flow solution can be used to efficiently solve other MRFs defined by *similar* energy functions.

Consider two MRFs M_a and M_b whose corresponding energy functions E_a and E_b differ by a few terms. As we have seen in the previous section this implies that the graph representing energy E_b i.e. G_b differs from that representing energy E_a i.e. G_a by a few edge costs. Suppose we have found the MAP solution of M_a by solving the max-flow problem on the graph G_a and now want to find the solution of M_b. Instead of the conventional procedure of recomputing the max-flow on G_b from scratch, we perform the computation by reusing the flows obtained while solving M_a.

Boykov and Jolly [4], in their work on interactive image segmentation used this technique for efficiently recomputing

[6]In our notation, if the node is disconnected from the source, we assign it the label zero and one otherwise.

the MAP solution when only the unary likelihood terms (8) change (due to addition of new hard and soft constraints by the user). However, they did not address the problem of handling changes in the pairwise terms of the energy function which result in changes in the cost of the n-edges of the graph. Our method (explained below) can handle arbitrary changes in the graph.

A. Updating Residual Graphs

The flow through a graph defines a residual graph (as explained in section II). Our algorithm works by updating the residual graph obtained from the max-flow computation in graph G_a to make it represent G_b. This is done by reducing or increasing the residual capacity of an edge according to the change made to its cost going from G_a to G_b.

While modifying the residual graph certain flows may violate the new edge capacity constraints(1). This is because flow in certain edges might be greater than the capacity of those edges under G_b. To make these flows consistent with the new edge capacities we reparameterize the updated graph (using reparameterizations described in the previous section) to make sure that the flows satisfy the edge capacity constraints (1) of the graph. The max-flow is then computed on this reparameterized graph. This gives us the st-mincut solution of graph G_b, and hence the MAP solution of MRF M_b.

We now show how the residual graph is transformed to make such flows consistent. We use the two graph transformations given in section IV to increase the capacities of edges in G_b in which the flow exceeds the true capacity. These transformations lead to a reparameterization of the graph G_b. We can then solve the max-flow on this graph to get the solution of the max-flow on G_b.

The various changes that might occur to the graph going from G_a to G_b can be expressed in terms of changes in the capacity of t-edges and n-edges of the graph. The methods for handling these changes will be discussed now. We use $c^{'}_{si}$ to refer to the new edge capacity of the edge (s,i). $r^{'}_{si}$ and $f^{'}_{si}$ are used to represent the updated residual capacity and flow of the edge (s,i) respectively.

1) Modifying t-edge Capacities: Our method for updating terminal or t-edges is similar to the one used in [4] and is described below.

The updated residual capacity of an edge (s,i) can be computed as: $r^{'}_{si} = r_{si} + c^{'}_{si} - c_{si}$. This can be simplified to: $r^{'}_{si} = c^{'}_{si} - f_{si}$. If the flow f_{si} is greater than the updated edge capacity $c^{'}_{si}$, it violates the edge capacity constraint (1) resulting in $r^{'}_{si}$ becoming negative. To make the flow consistent a constant $\gamma = f_{si} - c^{'}_{si}$ is added to the capacity of both the t-edges $\{(s,i),(i,t)\}$ connected to the node i. As has been observed in section 2 and in [4], this transformation is an example of graph reparameterization which does not change the minimum cut (it's cost changes but not the cut itself). For an illustration see figure 2. The residual capacities thus become: $r^{'}_{si} = c^{'}_{si} - f_{si} + \gamma = 0$ and, $r^{'}_{it} = c_{it} - f_{it} + \gamma$, or $r^{'}_{it} = r_{it} - c^{'}_{si} + f_{si}$.

2) Modifying n-edge Capacities: We now describe how the residual graph is updated when n-edge capacities are changed. Observe that updating edge capacities in the residual graph is simple if the new edge capacity $c^{'}_{ij}$ is greater than or equal to the old edge capacity c_{ij}. This operation involves addition of extra capacity and thus the flow cannot become inconsistent. The updated residual capacity $r^{'}_{ij}$ is obtained as:

$$r^{'}_{ij} = r_{ij} + (c^{'}_{ij} - c_{ij}). \tag{17}$$

Even if $c^{'}_{ij}$ is less than c_{ij}, the procedure still remains trivial if the flow f_{ij} is less than the new edge capacity $c^{'}_{ij}$. This is due to the fact that the reduction in the edge capacity does not affect the flow consistency of the network i.e flow f_{ij} satisfies the edge capacity constraint (1) for the new edge capacity. The residual capacity of the edge can still be updated according to equation (17). The difference in this case is that $(c^{'}_{ij} - c_{ij})$ is negative and hence will result in the reduction of the residual capacity. In both these cases, the flow through the edge remains unchanged i.e. $f^{'}_{ij} = f_{ij}$.

The problem becomes complex when the new edge capacity $c^{'}_{ij}$ is less than the flow f_{ij}. In this case, f_{ij} violates the edge capacity constraint (1). To make f_{ij} consistent, we have to retract the excess flow $(f_{ij} - c^{'}_{ij})$ from the edge (i,j). At this point, the reader should note that a trivial solution for this operation would be to push back the flow through the augmenting path it originally came through. However such an operation would be extremely computationally expensive. We now show how we resolve this inconsistency in constant i.e. O(1) time.

The inconsistency arising from excess flow through edge (i,j) can be resolved by a single valid transformation of the residual graph. This transformation is the same as the one shown in figure 2 for obtaining graph G_2 from G, and does not change the st-mincut. It leads to a reparameterization of the residual graph which has non-negative residual capacity for the edge (i,j). The transformation involves adding a constant $\alpha = f_{ij} - c^{'}_{ij}$ to the capacity of edges $(s,i),(i,j)$, and (j,t) and subtracting it from the residual capacity of edge (j,i). The residual capacity r_{ji} of edge (j,i) is greater than the flow f_{ij} passing through edge (i,j). As α is always less than f_{ij} the above transformation does not make the residual capacity of edge (j,i) negative. The procedure for restoring consistency is illustrated in figure 5.

B. Complexity Analysis of Update Operations

Modifying an edge cost in the residual graph takes constant time. Arbitrary changes in the graph like addition or deletion of nodes and edges can be expressed in terms of modifying an edge cost. The time complexity of all such changes is O(1) except for deleting a node where the update time is O(k). Here k is the degree of the node to be deleted[7].

After the residual graph has been updated to reflect the changes in the MRF the augmenting path procedure is used to find the maximum flow. This involves repeatedly finding paths with free capacity in the residual graph and saturating them. When no such paths can be found i.e. the source and sink are

[7]The capacity of all edges incident on the node has to be made zero which takes O(1) time per edge.

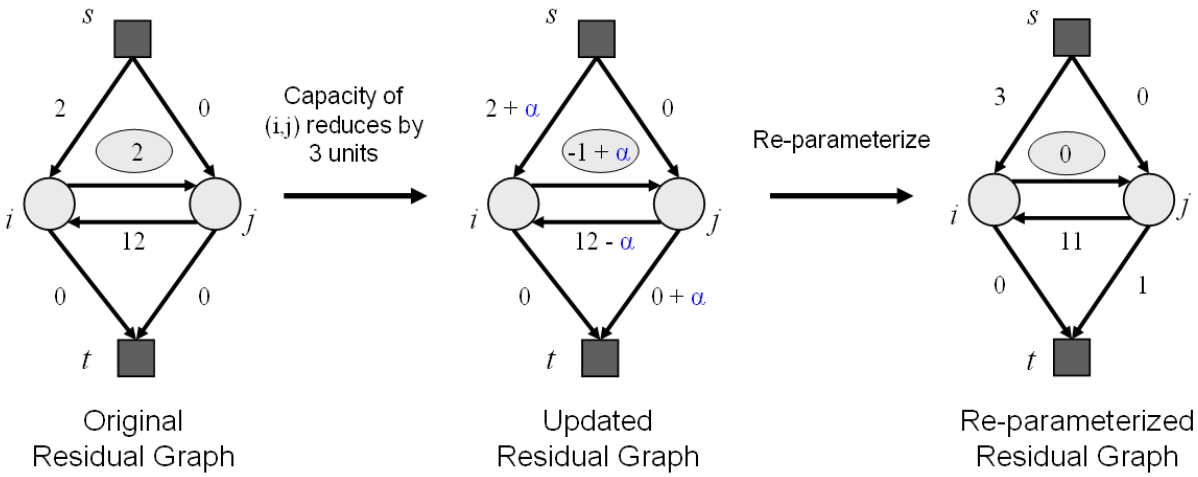

Fig. 5. *Restoring consistency using Graph Reparameterization. The figure illustrates how edge capacities can be made consistent with the flow by reparameterizing the residual graph. It starts by showing a residual graph consisting of two nodes i and j. obtained after a max-flow computation. For the second max-flow computation the capacity of edge (i, j) is reduced by 3 units resulting in the updated residual graph in which the residual capacity of edge (i, j) is equal to -1. To make the residual capacities positive we reparameterize the graph by adding $\alpha = 1$ to the capacity of edges (i, j), (s, i) and (j, t) and subtracting it from the capacity of edge (j, i). This gives us the reparameterized residual graph in which the edge flows are consistent with the edge capacities.*

disconnected in the residual graph, we reach the maximum flow.

The maximum flow from the source to the sink is an upper bound on the number of augmenting paths found by the augmenting path procedure. Also, the total change in edge capacity bounds the increase in the flow ∇f defined as:

$$\nabla f \leq \sum_{i=1}^{m^{'}} |c_{e_i}^{'} - c_{e_i}|, \qquad \text{where} \quad e_i \in E$$

or, $\nabla f \leq m^{'} c_{\max}$ where $c_{\max} = \max(|c_{e_i}^{'} - c_{e_i}|)$. Thus we get a *loose* O($m^{'} c_{\max}$) bound on the number of augmentations, where $m^{'}$ is the number of edge capacity updates.

VI. Optimizing the Algorithm

We have seen how by dynamically updating the residual graph we can reduce the time taken to compute the st-mincut. We can further improve the running time by using a technique motivated by [5].

Typical augmenting path based methods start a new breadth-first search for (source to sink) paths as soon as all paths of a given length are exhausted. For instance, Dinic [10] proposed an augmenting path algorithm which builds search trees to find augmenting paths. This is a computationally expensive operation, as it involves visiting almost all nodes of the graph, and makes the algorithm slow if it has to be performed too often. To counter this, Boykov and Kolmogorov [5] proposed an algorithm in which they reused the search tree. In their experiments, this new algorithm outperformed the best-known augmenting-path and push-relabel algorithms on graphs commonly used in computer vision.

Motivated from their results we decided to reuse the search trees available from the previous max-flow computation to find the solution in the updated residual graph. This technique saved us the cost of creating a new search tree and made our algorithm substantially faster. The main differences between our algorithm and that of [5] are the presence of the tree restoration stage, and the dynamic selection of active nodes. We will next describe how the algorithm of [5] works and then explain how we modify it to recycle search trees for dynamic graph cuts.

A. Reusing Search Trees

The algorithm described in [5] maintains two non-overlapping search trees S and T with roots at the source s and the sink t respectively. In tree S all edges from each parent node to its children are non-saturated, while in tree T edges from children to their parents are non-saturated. The nodes that are not in S or T are called *free*. The nodes in the search trees S and T can be either *active* (can *grow* by acquiring new children along non-saturated edges) or *passive*. The algorithm starts by setting all nodes adjacent to the terminal nodes as *active*. The three basic stages of the algorithm are as follows:

a) **Growth Stage:** The search trees S and T are grown until they touch each other (resulting in an augmenting path) or all nodes become *passive*. The active nodes explore adjacent non-saturated edges and acquire new children from the set of free nodes which now become active. As soon as all neighbours of a given active node are explored the active node becomes passive. When an active node comes in contact with a node from the other tree an augmenting path is found.

b) **Augmentation Stage:** In this stage of the algorithm flow is pushed through the augmenting path found in the growth stage. This results in some nodes of the trees S and T becoming *orphans* since the edges linking them to their parents become saturated. At this point the sink and source and sink search trees have decomposed into forests.

Fig. 6. *Segmentation in Videos using user seeds. The first image shows one frame of the input video with user segmentation seeds (the back and white boxes). The image pixels contained in these boxes are used to learn histograms modelling foreground and background likelihoods. The second image shows the segmentation result obtained using these likelihoods with the method of [4]. The result contains a certain portion of the background wrongly marked as the foreground due to similarity in colour. This error in the segmentation can be removed by the user by specifying a hard constraint. This involves marking a set of pixel positions in the wrongly labelled region as background (shown as the checkered region in the second image). This constraint is used for all the frames of the video sequence. The third image is the final segmentation result.*

c) **Adoption Stage:** During the adoption stage the search trees are restored by finding a new valid parent (of the same set) through a non-saturated edge for each orphan. If no qualifying parent can be found, the node is made free.

B. Tree Recycling for Dynamic Graph Cuts

We now explain our method for recycling search trees of the augmenting path algorithm. Our algorithm differs from that of [5] in the way we initialize the set of active nodes and in the presence of the Tree restoration stage.

1) Tree Restoration Stage: While dynamically updating the residual graph (as explained in section V) certain edges of the search trees may become saturated and thus need to be deleted. This operation results in the decomposition of the trees into forests and makes certain nodes *orphans*. We keep track of all such edges and before recomputing the st-mincut on the modified residual graph restore the trees by finding a new valid parent for each of them. This process is similar to the adoption stage and works as follows.

The aim of the tree restoration stage is two fold. First to find parents for orphaned nodes, and secondly but more importantly, to make sure that the length of the path from the root node to all other nodes in the tree is as small as possible. This is necessary to reduce the time spent passing flow through an augmenting path. Note that longer augmenting paths would lead to a slower algorithm. This is because the time taken to update the residual capacities of the edges in the augmenting path during the augmentation stage is proportional to the length of the path.

The first objective of the restoration stage can be met by using the adoption stage alone. For the second objective we do the following: Suppose node i belonged to the source tree before the updates. For each graph node i which has been affected by the graph updates we check the residual capacities of its t-edges ((s,i) or (i,t)). We can encounter the following two cases:

1) $r_{si} \geq r_{it}$: The original parent of the node (in this case, the source (s)) is reassigned as the parent of the node.

Fig. 7. *Segmentation results of the human lame walk video sequence.*

2) $r_{si} < r_{it}$: The parent of the node is changed to the other terminal node 'sink' (t). This means that the node has now become a member of sink tree T. All the immediate child nodes of i are then made orphans as they had earlier belonged the source tree.

The reassignment of parents of updated nodes according to the above mentioned rules resulted in a moderate but significant improvement in the results.

2) Dynamic Node Activation: The algorithm of [5] starts by marking the set of all nodes adjacent to the terminal nodes as *active*. This set is usually large and exploring all its constituent nodes is computationally expensive. However this is necessary as an augmenting path can pass through any such node.

In the case of the dynamic st-mincut problem however, we can isolate a much smaller subset of nodes which need to be explored for possible augmenting paths. The key observation to be made in this regard is that all new possible augmenting paths are constrained to pass through nodes whose edges have undergone a capacity change. This results in a much smaller active set and makes the max-flow computation significantly faster. When no changes are made to the graph all nodes remain *passive* and thus our augmenting path algorithm for computing the max-flow takes no time.

VII. Applications and Experimental Results

Our dynamic algorithm for the st-mincut problem has been used for a number of problems [7], [15], [21]. Here we demonstrate its performance on the problem of image segmentation in videos. We provide quantitative results comparing its performance with the dual-search tree algorithm proposed in [5] which has been experimentally shown to be the fastest for several vision problems including image segmentation[8]. We refer this algorithm as *static* since it starts afresh for each problem instance.

The dynamic algorithm which reuses the search trees will be referred to as the *optimized* dynamic graph cut algorithm. It should be noted that while comparing running times the time taken to allocate memory for graph nodes was not considered. Further, to make the experimental results invariant to cache performance we kept the graphs in memory.

A. Image Segmentation in Videos

The object-background segmentation problem aims to cut out user specified objects in an image [4]. We consider the

[8]For the static algorithm we used the authors original implementation.

case when this process has to be performed over all frames in the video sequence. The problem is formulated as follows.

The user specifies hard and soft constraints on the segmentation by providing segmentation cues or seeds on only the first frame of the video sequence. The **soft constraints** are used to build colour histograms for the *object* and *background*. These histograms are later used for calculating the likelihood term $\phi(\mathbf{D}|f_i)$ of the energy function (11) of the MRF [4] for all the frames of the video sequence.

The **hard constraints** are used for specifying pixel positions which are constrained to take a specified label (*object* or *background*) in all the frames of the video sequence. Note that unlike soft constraints, the pixel positions specified under hard constraints do not contribute in the construction of the colour histograms for the *object* and *background*. This is different from the user-input strategy adopted in [4]. In our method the hard constraints are imposed on the segmentation by incorporating them in the likelihood term $\phi(\mathbf{D}|f_i)$. This is done by imposing a very high cost for a label assignment that violates the hard constraints in a manner similar to [4]. This method for specifying hard constraints has been chosen because of its simplicity. Readers should refer to [35] for a sophisticated method for specifying hard constraints for the video segmentation problem. Figure 6 demonstrates the use of constraints in the image segmentation process. The segmentation results are shown in figure 7.

B. Experimental Results

The video sequences used in our tests had between one hundred to a thousand image frames. For all the video sequences dynamically updating the residual graph produced a decrease in the number of augmenting paths. Further the dynamic algorithms (normal and optimized) were substantially faster than the *static* algorithm. The average running times per image frame for the static, dynamic and optimized-dynamic algorithms for the human lame walk sequence[9] of size (368x256) were 91.4, 66.0, and 33.6 milliseconds and for the grazing cow sequence of size (720x578) were 188.8, 151.3, and 78.0 milliseconds respectively. The time taken by the dynamic algorithm includes the time taken to recycle the search trees. The experiments were performed on a Pentium 4 2.8 GHz machine.

The graphs in figure 8 show the performance of the algorithms on the first sixty frames of the human lame walk sequence. Observe that the number of augmenting paths found is lowest for the dynamic algorithm, followed by the dynamic (optimized) and then the static algorithm. This difference is due to the use of recycled search trees in the optimized algorithm.

VIII. Reusing Flow Vs Reusing Search Trees

In this section, the relative contributions of reusing flow and search trees in improving the running time of the dynamic algorithm are discussed.

The procedure for constructing a search tree has linear time complexity and thus should be quite fast. Further as seen in figure 8 using a fresh search tree after every graph update results in fewer augmenting paths. From these results it might appear that recycling search trees would not yield a significant improvement in running time. However this is not the case in practice as seen in figure 9. This is because although the complexity of search tree construction is linear in the number of edges in the graph, the time taken for tree construction is still substantial. This is primarily due to the nature of graphs used in computer vision problems. The number of nodes/edges in these graphs may be of the order of millions. For instance, when segmenting an image of size 640×480, max-flow on a graph consisting of roughly 3×10^5 nodes and more than 2 million edges needs to be computed. The total time taken for this operation is 90 milliseconds (msec) out of which almost 15 msec is spent on constructing the search tree.

The time taken by the dynamic algorithm to compute the st-mincut decreases with the decrease in the number of changes made to the graph. However, as the time taken to construct the search tree is independent of the number of changes, it remains constant. This results in a situation where if only a few changes to the graph are made (as in the case of min-marginal computation [21]), the dominant part of computation time is spent on constructing the search tree itself. By reusing search

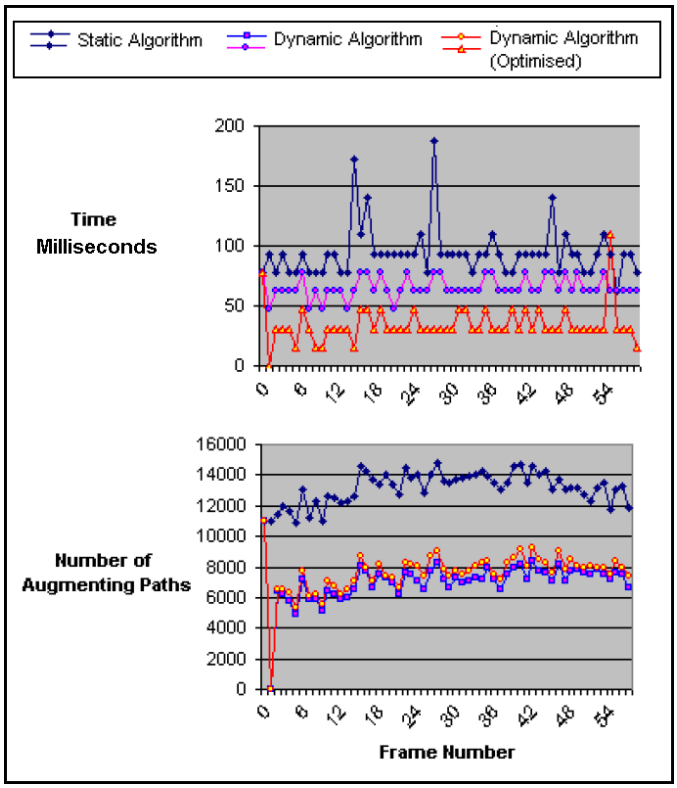

Fig. 8. *Running time and number of augmenting paths found by the different algorithms. Observe as the first and second frames of the video sequence are the same, the residual graph does not need to be updated, which results in no augmenting paths found by the dynamic algorithms when segmenting frame 2. Further, the optimized dynamic algorithm takes no time for computing the segmentation for the second image frame as the* MRF*s corresponding to the first and second image frames are the same and thus no modifications were needed in the residual graph and search trees. However, the normal dynamic algorithm takes a small amount of time since it recreates the search trees for every problem instance from scratch.*

[9]Courtesy Derek Magee, University of Leeds.

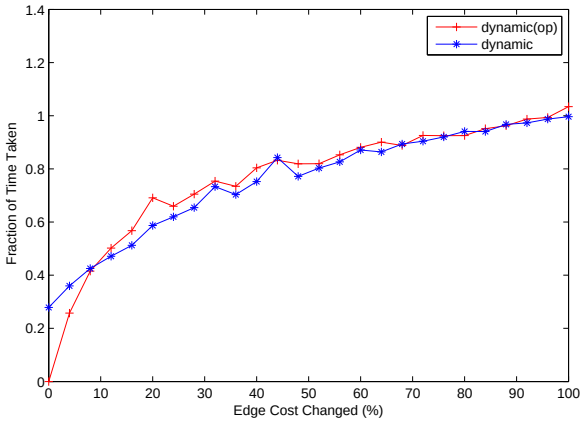

Fig. 9. *Behavior of the dynamic algorithm. The figure illustrates how the time taken by the dynamic algorithm (with/without tree recycling) changes with the number of modifications made to the graph. The graph shows the fraction of time taken to compute the st-mincut in the updated residual graph (with/without tree recycling) compared to that taken for computing the st-mincut in the original graph using the algorithm of [5]. For this experiment, we used a graph consisting of* $1x10^5$ *nodes which were connected in a 8-neighbourhood. The dynamic algorithm with tree recycling is referred as dynamic(op).*

trees we can get rid of this constant cost of creating a search tree and replace it with a change dependent tree restoration cost.

The exact amount of speed-up contributed by reusing flow and search trees techniques varies with the problem instance. For a typical interactive image segmentation example, the first st-mincut computation takes 120 msec out of which 30 msec is spent on constructing the search tree. We need to recompute the st-mincut after further user interaction (which results in changes in the graphs). For the later st-mincut computation, if we construct a new search tree then the time taken by the algorithm is 45 msec (a speed up of roughly 3 times) out of which 30 msec is used for tree creation and 15 msec is used for flow computation. However, if we use reuse the search trees, then the algorithm takes only 25 msec (a speed up of 5 times) out of which 7 msec is used for recycling the tree and 18 msec is used for flow computation.

Our results indicate that when a small number of changes are made to the graph the recycled search tree works quite well in obtaining short augmenting paths. The time taken for recycling search trees is also small compared to the time taken to create a new search tree in a large graph. With increased change in the graph the advantage in using the recycled search tree fades due to the additional number of flow augmentations needed as a result of longer augmentation paths obtained from the search tree.

IX. Conclusion

In this paper we presented a new *fully* dynamic algorithm for the st-mincut problem which can be used to find MAP solutions for certain dynamically changing MRFs rapidly. It should be noted that our method is generic and finds exact solutions for all dynamic problems which can be formulated as sub-modular energy functions of binary variables. The results show that our algorithm is substantially faster than the best known static st-mincut algorithm. We have demonstrated how our method can be used to perform efficient image segmentation in video sequences in a manner much faster than previously possible.

Acknowledgment

We thank associate editor Prof. Ramin Zabih and the anonymous reviewers for suggestions that helped us improve the presentation of the paper. This work was supported by the EPSRC research grant GR/T21790/01(P) and the IST Programme of European Community, under the PASCAL Network of Excellence.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. Network flows, 1993.
[2] A. Blake, C. Rother, M. Brown, P. Perez, and P. H. S. Torr. Interactive image segmentation using an adaptive gmmrf model. In *ECCV04*, pages Vol I: 428–441, 2004.
[3] E. Boros and P. L. Hammer. Pseudo-boolean optimization. *Discrete Applied Mathematics*, 123(1-3):155–225, 2002.
[4] Y. Boykov and M. P. Jolly. Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images. In *ICCV*, pages I: 105–112, 2001.
[5] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *PAMI*, 26(9):1124–1137, September 2004.
[6] Y. Boykov, O. Veksler, and R. Zabih. Markov random fields with efficient approximations. In *CVPR*, pages 648–655, 1998.
[7] M. Bray, P. Kohli, and P. H. S. Torr. Posecut: Simultaneous segmentation and 3d pose estimation of humans using dynamic graph cuts. In *ECCV*, pages 642–655, 2006.
[8] Y. Chiang and R. Tamassia. Dynamic algorithms in computational geometry. In *IEEE Special Issue on Computational Geometry*, volume 80, pages 362–381, 1992.
[9] R. F. Cohen and R. Tamassia. Dynamic expression trees and their applications. In *SODA*, pages 52–61, 1991.
[10] E. A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Math. Dokl.*, 11:1277–1280, 1970.
[11] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, 1962.
[12] D. Freedman and P. Drineas. Energy minimization via graph cuts: Settling what is possible. In *CVPR (2)*, pages 939–946, 2005.
[13] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM J. on Comput.*, 18:18:30–55, 1989.
[14] D. Greig, B. Porteous, and A. Seheult. Exact maximum a posteriori estimation for binary images. *RoyalStat*, B: 51(2):271–279, 1989.
[15] A. Hengel, A. Dick, T. Thormhlen, B. Ward, and P. H. S. Torr. Rapid interactive modelling from video with graph cuts. In *Eurographics*, 2006.
[16] H. Ishikawa. Exact optimization for markov random fields with convex priors. *PAMI*, 25(10):1333–1336, October 2003.
[17] H. Ishikawa and D. Geiger. Occlusions, discontinuities, and epipolar lines in stereo. In *ECCV (1)*, pages 232–248, 1998.
[18] H. Ishikawa and D. Geiger. Segmentation by grouping junctions. In *CVPR*, pages 125–131, 1998.
[19] O. Juan and Y. Boykov. Active graph cuts. In *CVPR (1)*, pages 1023–1029, 2006.
[20] P. Kohli and P. H. S. Torr. Efficiently solving dynamic markov random fields using graph cuts. In *ICCV*, pages 922–929, 2005.
[21] P. Kohli and P. H. S. Torr. Measuring uncertainty in graph cut solutions: Efficiently computing min-marginal energies using dynamic graph cuts. In *ECCV*, pages 30–43, 2006.
[22] V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1568–1583, 2006.

[23] V. Kolmogorov and Y. Boykov. What metrics can be approximated by geo-cuts, or global optimization of length/area and flux. In *ICCV*, pages 564–571, 2005.
[24] V. Kolmogorov and R. Zabih. Multi-camera scene reconstruction via graph cuts. In *ECCV (3)*, pages 82–96, 2002.
[25] V. Kolmogorov and R. Zabih. What energy functions can be minimized-via graph cuts?. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(2):147–159, 2004.
[26] M. P. Kumar, P. H. S. Torr, and A. Zisserman. Learning layered pictorial structures from video. In *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing*, pages 158–163, 2004.
[27] M. P. Kumar, P. H. S. Torr, and A. Zisserman. Obj cut. In *CVPR05*, pages 18–25, 2005.
[28] A. Raj and R. Zabih. A graph cut algorithm for generalized image deconvolution. In *ICCV*, pages 1048–1054, 2005.
[29] C. Rother, V. Kolmogorov, and A. Blake. "grabcut": interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.*, 23(3):309–314, 2004.
[30] S. Roy and I. J. Cox. A maximum-flow formulation of the n-camera stereo correspondence problem. In *ICCV*, pages 492–502, 1998.
[31] M. I. Schlesinger and B. Flach. Some solvable subclasses of structural recognition problems. In *Czech Pattern Recognition Workshop*, 2000.
[32] D. Snow, P. Viola, and R. Zabih. Exact voxel occupancy with graph cuts. pages I: 345–352, 2000.
[33] M. Thorup. Fully-dynamic min-cut. In *STOC*, pages 224–230, 2001.
[34] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky. Map estimation via agreement on trees: message-passing and linear programming. *IEEE Transactions on Information Theory*, 51(11):3697–3717, 2005.
[35] J. Wang, P. Bhat, A. Colburn, M. Agrawala, and M. F. Cohen. Interactive video cutout. *ACM Trans. Graph.*, 24(3):585–594, 2005.
[36] J. Xiao and M. Shah. Motion layer extraction in the presence of occlusion using graph cut. In *CVPR (2)*, pages 972–979, 2004.
[37] Stan Z. Li. *Markov Random Field Modeling in Computer Vision*. Springer-Verlag, 2001.
[38] R. Zabih and V. Kolmogorov. Spatially coherent clustering using graph cuts. In *CVPR (2)*, pages 437–444, 2004.

Pushmeet Kohli received his BTech degree in Computer Science and Engineering from the National Institute of Technology, Warangal in 2004. He is currently a PhD student at Oxford Brookes University, Oxford and is being funded by a EPSRC studentship. His research interests include information theory, machine learning and computer vision. He has worked as a research intern for Microsoft Research in the Foundations of Software Engineering (FSE) and Machine learning and Perception (MLP) groups.

Philip H. S. Torr did his PhD (DPhil) at the Robotics Research Group of the University of Oxford under Professor David Murray of the Active Vision Group. He worked for another three years at Oxford as a research fellow. He left Oxford to work for six years as a research scientist for Microsoft Research, first in Redmond USA in the Vision Technology Group, then in Cambridge UK founding the vision side of the Machine learning and perception group. He is now a Professor in Computer Vision and Machine Learning at Oxford Brookes University. Philip Torr won the Marr prize in 1998. He was involved in the algorithm design for Boujou released by 2D3. Boujou has won a clutch of industry awards, including Computer Graphics World Innovation Award, IABM Peter Wayne Award, and CATS Award for Innovation, and an EMMY. He is a senior member of the IEEE and on the editorial boards of ACM Computers and Entertainment, IEEE Transactions on Pattern Analysis and Machine Intelligence and the Journal of Image and Vision Computing.