An Efficient Graph Cut Algorithm for Computer Vision Problems

Chetan Arora, Subhashis Banerjee, Prem Kalra, and S.N. Maheshwari

Department of Computer Science and Engineering,
Indian Institute of Technology, Delhi, India
{chetan,suban,pkalra,snm}@cse.iitd.ac.in

Abstract. Graph cuts has emerged as a preferred method to solve a class of energy minimization problems in computer vision. It has been shown that graph cut algorithms designed keeping the structure of vision based flow graphs in mind are more efficient than known strongly polynomial time max-flow algorithms based on preflow push or shortest augmenting path paradigms [1]. We present here a new algorithm for graph cuts which not only exploits the structural properties inherent in image based grid graphs but also combines the basic paradigms of max-flow theory in a novel way. The algorithm has a strongly polynomial time bound. It has been bench-marked using samples from Middlebury [2] and UWO [3] database. It runs faster on all 2D samples and is at least two to three times faster on 70% of 2D and 3D samples in comparison to the algorithm reported in [1].

1 Introduction

Many problems in computer vision such as image segmentation [4], stereo [5], texture synthesis [6], multi-view reconstruction [7] have been modelled as label assignment problems involving energy minimization. Label assignment problem is NP hard in general [8]. However, for a number of problems (e.g. texture synthesis, segmentation etc.) the label set has only two labels. It is well known that in the two label case the energy minimization problem can be modelled as determining a minimum capacity cut in a flow graph [9]. Two label case is also important because many multiple labelling algorithms use binary labelling repeatedly to get to an acceptable solution [8,10]. Graph cuts are also used to solve MAP (maximum a *posteriori*) solution for discrete MRFs. Apart from efficient algorithms for determining graph cuts [1,11,12,13], recent research has focussed on mapping computer vision problems on appropriately constructed graphs [6,10,14,15] and characterizing energy functions that can be minimized by graph cuts [8,10].

Finding a minimum cut in a flow graph is equivalent to solving the max-flow in it and [11,12,13] focus on implementing/adapting known polynomial time max-flow algorithms to run on flow graphs obtained from vision problems. Boykov and Kolmogorov [1] have developed a max-flow algorithm specifically with the objective of practical efficiency when run on such flow graphs. They included a study of comparative performance of their algorithm with the standard shortest augmenting path based algorithm [16], and variations of the preflow push algorithm [17,18]. They [1] showed that while the provable time bound (for integer capacities) of their algorithm (referred to as BK from now on) was weaker ($O(n^3C)$, where n is the number of nodes and C is the cost of minimum cut in comparison to $O(n^3)$ for the standard algorithms), their algorithm out-performed the others in practice. Goldberg [19] compares experimental performance of BK with some variations of preflow push that have been proposed since [1]. BK continues to out-perform the others on most and particularly the two dimensional data sets.

Contribution: We present in this paper a new flow based graph cut algorithm which is both strongly polynomial and efficient in practice. We show that in comparison to our new algorithm the best known algorithm [1] is slower by a factor of 2 to 3 on most of the BVZ (2D dataset), bone, bunny, babyface and adhead (3D datasets) samples from the UWO database [3]. On the Middlebury database [2] used by [20] to test their GPU implementation, we show that our algorithm is 3 times faster than the time reported for the GPU runs. At a very macro level our algorithm may be viewed as a hybrid of the preflow push strategy with the layered graph approach of augmenting path methods. The algorithm keeps the simplicity and locality of preflow strategies while at same time borrows ideas from layered graph based augmenting path methods to give general directions to flow.

Section 2 describes our algorithm. In Section 3 we present results and comparison with currently known best methods in the field followed by conclusion and discussion in Section 4.

2 Voronoi Based Preflow Push (VPP)

We first review some of the basic terminology we use. We assume that the node set of original grid graph is augmented by two additional vertices s and t called source and sink respectively. The edge set of such graph (called *flow graph* hereinafter) consists of all the neighbourhood edges called n-links, and t-links which connect s and t to all nodes in N (the edges are directed from s to nodes in N, and from nodes in N to t). We assume that between two nodes p and q both directed edges (p, q) and (q, p) exist. Each n-link (p, q), and t-link connecting a node $p \in N$ to s and t has capacity greater than or equal to zero (V_{pq} denotes capacity of edge (p, q)). An (S, T) cut in this flow graph is defined as a partitioning of the nodes into sets S and T such that s is in S and t is in T. Capacity of an (S, T) cut is the sum of the capacities of edges directed from S to T. Flow in a flow graph is a non negative real valued function that associates a value f_{pq} with an edge (p, q) in the flow graph where $f_{pq} \leq V_{pq}$. *Effective flow* in edge (p, q) is, therefore, equal to f_{pq} - f_{qp}. *Residual capacity* of an edge (p, q), denoted by $residue(p, q)$, is a measure of the additional flow that can be sent through it in the presence of some existing flow. An edge with non-zero residual capacity is called a *residual edge*. *In-flow/Out-flow* at node is the sum of the effective flow in all the edges directed into/out of node. A flow is a *preflow* if in-flow is at least as large as out-flow at all nodes other than the source and the sink. *Excess(v)* of a node v is equal to in-flow minus out-flow at node v. Consider a starting configuration in which flow is set equal to the capacity in t-links and equal to zero in n-links. Now if we label the vertices other than s and t by their excesses, it is easy to see that the original max-flow problem is equivalent to finding max-flow in the flow graph in which all t-links have been removed. Source-Sink max-flow problem is solved in the resultant flow graph by treating nodes with positive excesses as sources and those with negative excesses as sinks. From now on we will assume that the max-flow problem is being solved on such a flow graph. It should be noted that in this version of the problem sources and sinks do not have unlimited capacity, rather they have the ability to send or absorb only the amount equal to the excess on them. With every node v of the grid graph we associate label $d(v)$ called *distance label* (or simply *label*) satisfying the following conditions: $d(v) = 0$ for all nodes with negative excesses and for every residual edge (v, w), $d(v) \leq d(w) + 1$. A residual edge (v, w) will be an out-edge/in-edge of v/w if $d(v) = d(w) + 1$.

Broad steps of our algorithm (referred to as *VPP* hereafter) are given in Algorithm 1. Our algorithm differs from standard preflow push based algorithms in some crucial ways.

Algorithm 1. Voronoi Based Preflow Push

```
1: Create shortest distance based Voronoi region graphs around sink clusters;
2: for Voronoi region graphs with sources do
3:     Push flow from the sources towards the sink cluster in each such Voronoi region followed
       by pushing flow within the sink cluster;
4:     rebuild the Voronoi region graphs around remaining sink clusters;
5: end for
```

1. Unlike preflow push algorithms we maintain exact distance labels from the sinks. Note that in computer vision problems, sources and sinks are very often *clustered* (collection of source(sink) nodes in which there is a path between any two nodes passing through only nodes in the collection is called a *cluster*). Also such clusters are often interspersed. In *VPP* distance labels are shortest distance to a sink node on a sink cluster boundary. Initially these labels are generated by the standard global labelling procedure of push relabel algorithms and at later iterations by an *incremental relabelling algorithm* developed specifically to control the number of nodes relabelled at each iteration. Assignment of distance labels stops once all the sources are labelled. The subgraphs of nodes and their in and out edges reachable from a sink cluster are called *Voronoi region graph*.
2. Flow is pushed in a push flow iteration by processing the nodes in topological sort order (similar to the *highest label first* heuristic in preflow push algorithms). This ensures that once flow is pushed out of a node flow will not be pushed in it in the current push iteration. There is no local relabelling step at all.
3. Once flow reaches the boundary nodes of a sink cluster the second phase of the push iteration is initiated. This consists of starting from all the boundary nodes of a sink cluster and pushing flow inwards in the sink cluster by processing the cluster nodes in a breadth first manner till either all the excess gets absorbed among the nodes of the cluster or there are no nodes left in the cluster. The first case will result in a smaller sink cluster(s) and in the second the sink cluster will disappear. In push relabel based algorithms when a boundary sink node of a cluster gets saturated, the inside neighbour in the cluster gets exposed. This results in distance labels of

a large set of nodes to be recomputed. Pushing flow within a sink cluster before relabelling contributes to pushing as much flow as possible towards sink clusters between two relabelling steps.

Maintaining exact distance labels and the flow pushing strategy used ensures that flow that gets pushed into a sink cluster originates only from sources that lie on the Voronoi region graphs associated with the sink cluster. Also, because flow pushing to all sink clusters takes place simultaneously between two relabelling iterations, changes in Voronoi boundaries is incremental as long as sink clusters do not disappear. In effect most of the time flow pushing towards a sink cluster takes place within that part of the grid graph that became part of the Voronoi region graph during initialization. Normally only after a cut has been discovered and/or a sink cluster has disappeared will sources change their association and become part of other Voronoi region graphs. Such changes in association contribute significantly to the relabelling cost. Pushing flow within Voronoi regions, therefore, works to control relabelling costs. Preflow push algorithms with local relabelling cannot focus on this issue as implicit Voronoi boundaries can change with every relabel. At the surface level algorithms like HI_PR [18] which use variations of highest level first push strategy with occasional global relabelling (the process of creating exact distance labels) seem to be very similar. However, we show in section 3 that the number of nodes touched in HI_PR in both pushing flow and relabelling phases is an order of magnitude larger than our algorithm. This is primarily due to: (i) there are no wasteful local relabelling steps: flow pushing takes place only within Voronoi region graphs, (ii) repeated relabelling caused by shrinking cluster boundaries is avoided, and (iii) incremental relabelling process, explained in detail in a following section, ensures efficient calculation of exact distance labels.

Details of steps at line number 1, 3, and 4 of Algorithm 1 are provided in the following sections.

2.1 Initialization

We call the process of creating shortest distance based Voronoi region graphs as the initialization phase. This process is similar to the global relabelling phase of a traditional preflow push algorithm starting with initializing all sinks at distance label 0. The Voronoi region graphs with in/out edge lists at every node get created. $Excess(v)$ is initialized to 0 for all nodes v other than sources and sinks. For all source and sink nodes $excess(v)$ is set equal to source capacity if v is a source or equal to negative of sink capacity if v is a sink. Initialization also inserts all sources in structures called *Excess List (EL)*. Excess lists are maintained for each distance label. $EL(d)$ contains source nodes with distance label d. d_{max} is the largest distance label assigned to any node.

2.2 Push Flow

Push flow happens in two phases. First phase (Algorithm 2) takes excess from sources to the boundaries of sink cluster following the highest distance label first strategy. At a node flow is pushed saturating the out edges till the node has no excess left or all out edges of the node get saturated. The saturated out edges are deleted and a node whose all out-edges are deleted is inserted in a list called *Disconnected List(DL)*. Pushing flow may also involve inserting the node into which flow is being pushed into an EL and deleting the node whose excess becomes zero from an EL. DLs are maintained for each distance label d. Second phase moves the excess that accumulates at boundary nodes of a sink cluster inside the sink cluster. Second phase (Algorithm 4) starts from those sink cluster boundary nodes with positive excess on them and pushes the excess inside the cluster in a breadth first manner.

Algorithm 2. Voronoi Push Flow Phase 1

```
1: for $d = d_{max}..1$ do
2:    for all $v$ in $EL(d)$ do
3:       while $excess(v) > 0$ and $v$ has out-edges do
4:          pick an out-edge (v,w);
5:          Push_Flow_in_Edge (v,w);
6:          insert $w$ in $EL(d(w))$ if not already inserted;
7:          if $residue(v,w) = 0$, delete edge (v,w) from out-edges and in-edges of $v$ and $w$
            respectively;
8:       end while
9:       delete $v$ from $EL(d)$;
10:      if all out-edges of $v$ have been deleted, then insert $v$ in $DL(d)$;
11:   end for
12: end for
```

Procedure 3. Push_Flow_in_Edge (v,w)

```
1: $f \leftarrow min(excess(v), residue(v,w))$;
2: $excess(v) \leftarrow excess(v) - f$;
3: $excess(w) \leftarrow excess(w) + f$;
4: $residue(v,w) \leftarrow residue(v,w) - f$;
5: $residue(w,v) \leftarrow residue(w,v) + f$;
```

2.3 Rebuilding the Acyclic Voronoi Regions

A node v is labelled disconnected during the Push flow stage because all paths from v to the sink of a Voronoi region graph have been saturated and there is no remaining path from v to a sink in the Voronoi region graph on which flow can be pushed. Specifically, these are nodes put in $DL(d)$ in step 10 of Algorithm 2. It is important to note here that these nodes are not all the nodes for which there do not exist augmenting paths in the Voronoi region graphs after the Push flow stage. Other nodes in the Voronoi region graphs for which all paths to a sink pass through nodes put in $DL(d)$ are also effectively disconnected. Algorithm 5 identifies all such additional nodes (step 5) and adds them to the $DL(d)$. Other nodes (i.e. nodes not put in $DL(d)$ in step 10 of Algorithm 2 or step 5 of Algorithm 5) continue to have augmenting paths to sinks in the Voronoi region graphs and hence have the correct shortest distance label. An augmenting path from a node v in $DL(d)$ to a sink in the new residual graph will necessarily have to

Algorithm 4. Push Flow Phase 2

```
1: for all sinks $v$ set $BfsLevel(v) = \infty$;
2: for all $v$ in EL(0) do
3:     if $excess(v) \geq 0$ then
4:         $BfsLevel(v) = 0$
5:         insert $v$ in CurrentBfsList
6:     end if
7: end for
8: while CurrentBfsList is not empty do
9:     for all $v$ in CurrentBfsList do
10:        while any (v,w) with $residue(v,w) > 0$ exists and $excess(v) > 0$ do
11:            if w is a sink with $BfsLevel(w) > BfsLevel(v)$ then
12:                Push_Flow_in_Edge (v,w);
13:                if $BfsLevel(w) = \infty$ then $BfsLevel(w) \leftarrow BfsLevel(v) + 1$;
14:                insert $w$ in NextBfsList if not already inserted;
15:            end if
16:        end while
17:        if $excess(v) \geq 0$ then insert $v$ in $DL(0)$;
18:        delete $v$ from CurrentBfsList;
19:    end for
20:    swap CurrentBfsList and NextBfsList;
21: end while
```

pass through a node which continues to retain its shortest distance label after a push flow stage. Also, such a path, if it exists, from a node v, whose all out edges have been saturated during push flow, will have to pass through a neighbour w not in its out-edge list as it existed when flow was pushed last. For such a node w, $d(w)$ was greater than or equal to $d(v)$ in the Voronoi region graph in which flow was pushed, and so the new label for v will be larger than its current label. This also implies that the label of all those nodes u, for which augmenting paths to the sink pass through v in the new residual graph, either increase their labels as well or the edge *(u,v)* be dropped from the Voronoi region graph to retain consistency among distance labels. We give below a two phase incremental relabelling process the first phase of which (Algorithm 5) identifies nodes whose shortest distance labels will increase (the disconnected nodes added to $DL(d)$s) and those which have residual edges pointing to them from a newly discovered disconnected node (inserted in a *Rebuild List (RL)*). In first phase the DL lists are processed in order of increasing distance labels thereby ensuring that at the end of processing nodes in $Dl(d)$, the $DL(d+1)$ and $RL(d)$ lists have been correctly computed. Nodes in $RL(d)$ can provide distance label $d+1$ to disconnected nodes.

To ensure that disconnected nodes get the shortest distance label, second phase (Algorithm 6) starts with the lowest level non empty rebuild list. It can be shown that after $RL(d)$ has been processed all disconnected nodes which are at shortest distance $d+1$ from a sink have been so labelled. Such nodes would necessarily have to have a residual edge directed from them to a node whose shortest distance label is d. The phase one and two ensure that such nodes will be in $RL(d)$. It is possible that in the process of rebuilding, a node shifts from one Voronoi region to another. This will depend upon

Algorithm 5. Rebuild Phase 1

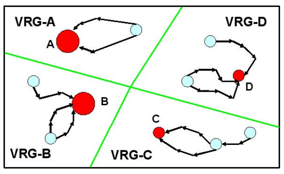
```
1: for $d = 0..d_{max}$ do
2:   for each $v$ in $DL(d)$ do
3:     for all edges (w,v) do
4:       remove edge (w,v) from out-edges and in-edges of $w$ and $v$ respectively;
5:       if there are no out-edges in $w$ then insert it in $DL(d(w))$;
6:       if $w$ exists in $RL(d(w))$ then delete it from $RL(d(w))$;
7:     end for
8:     for all residual edges (v,u), if u has any out-edge, then insert u in RL(d(u)) if not already
       inserted;
9:     $d(v) \leftarrow \infty$;
10:    delete $v$ from $DL(d)$;
11:  end for
12: end for
```

the Voronoi region to which the node in the rebuild list through which the disconnected node finds a new path to a sink belonged.

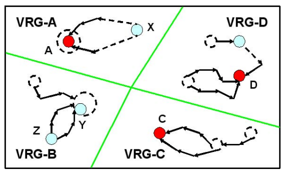
Algorithm 6. Rebuild Phase 2

```
1: for $d = 0..d_{max}$ do
2:   for each $v$ in $RL(d)$ do
3:     for all edges (w,v) with $residual(w, v) > 0$ do
4:       if $d(w) = d + 1$ then
5:         make $(w, v)$ an out-edge of $w$ and in-edge of $v$;
6:       else
7:         if $d(w) = \infty$ then
8:           $d(w) \leftarrow d + 1$;
9:           if $d_{max} < d(w)$ then $d_{max} \leftarrow d(w)$;
10:          if $excess(w) > 0$ then insert $w$ in $EL(d(w))$
11:          make $(w, v)$ an out-edge of $w$ and in-edge of $v$;
12:          insert $w$ in $RL(d(w))$;
13:        end if
14:      end if
15:    end for
16:    delete $v$ from $RL(d)$;
17:  end for
18: end for
```

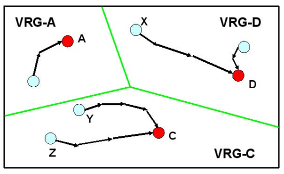
Figures 1(a), 1(b), and 1(c) depict the state of the flow graph prior to a push flow iteration, after the push flow iteration, and after the corresponding rebuild phase. In these figures sinks clusters are circles labelled A,B,C, and D. Rest of the circles are source clusters. Figure 1(a) represents a possible scenario prior to a push phase with four Voronoi regions corresponding to the four sink clusters. Directed lines are parts of the shortest paths that exist in the Voronoi region graphs. Flow will get pushed in each of these four Voronoi region graphs starting from the furthest away sources in topological



(a) After initialization/rebuild



(b) After Push flow



(c) After rebuilding Voronoi regions

Fig. 1. Flow graph states in VPP

sort order. Figure 1(b) represents the state after the push flow phase. Thick dashed lines indicate saturated edges. Dashed circles show the changes in sources and sink clusters. Note that in region VRG-A the sink cluster has shrunk, in VRG-B the sink cluster has disappeared and a new source created (circle labelled Y). In VRG-C and VRG-D sink clusters have not changed but a few source clusters have disappeared and new ones created. Figure 1(c) represents the state of the flow graph immediately after the rebuild phase. The three Voronoi regions correspond to the remaining sink clusters. Voronoi boundaries have shifted and sources X, Y, and Z are now in different Voronoi regions.

The algorithm finds the value of the max flow and the corresponding minimum energy graph cut. Flow in the graph may be a preflow when the algorithm terminates. The standard phase of converting a preflow into a flow would need to be incorporated to convert the above into a max flow algorithm [18].

Worst case time complexity of the algorithm is easily established. Beyond the initialization phase an iteration of the algorithm involves pushing flow and rebuilding the acyclic Voronoi regions. In a push flow phase, flow is pushed in an edge only once and so the total time taken is bounded by the number of edges in the acyclic Voronoi regions. In grid graphs with bounded degree, number of edges are $O(n)$. Rebuilding the acyclic Voronoi regions requires two passes over the grid graph in which edges and nodes of the graph that are accessed are touched a constant number of times. Rebuild time in each iteration is, therefore, $O(n)$. Shortest distance labels on nodes in the grid graph increase monotonically. Between iterations at least one node in the graph will have its label increased by one. Shortest distance labels can not remain the same between iterations as that would imply no change in the acyclic graph structure between iterations. This is not possible if there are nodes at the end of an iteration with positive excesses. Since there are n nodes in the grid graph the maximum number of iterations possible is $O(n^2)$ (under the assumption that only one node gets its labelled increased in any iteration and that the maximum label any node can have is n). Over all time complexity of the algorithm is, therefore, $O(n^3)$. We would like to mention that the above analysis simply establishes strong polynomial bound and is not necessarily the sharpest bound provable. However, we show experimentally in section 3 that the actual number of iterations on vision grid graphs is much less. Tight analysis of the algorithm on vision grid graphs is an open issue.

3 Results and Comparison

We have implemented our algorithm in C++ and compared its performance with BK [21], and HI_PR [22] on a machine with dual core 2.5 GHz CPU with 2GB RAM. Performance comparison with P2R [19] and CH-n [23] has been included on the basis of results on 3D datasets in [19]. Comparison with CudaCuts is using times reported in [20] on data sets from Middlebury [2]. 2D samples are BVZ and 3D samples are bunny, babyface, adhead and bone from UWO database [3]. For tests that we have conducted we have measured only time taken to run the algorithms after the flow graph has been constructed. Accuracy of the algorithm is verified by matching the flow computed by it to the one given with the database.

Figure 2 tabulates the time taken by our algorithm and CudaCuts time as reported in [20]. Our algorithm is 2 to 3 times faster.

Sample	VPP Time (ms)	CudaCuts Time (ms)
flower	19.81	37
person	20.77	61
sponge	17.74	44

Fig. 2. Time comparison between CudaCuts and VPP

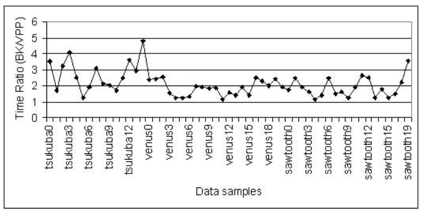
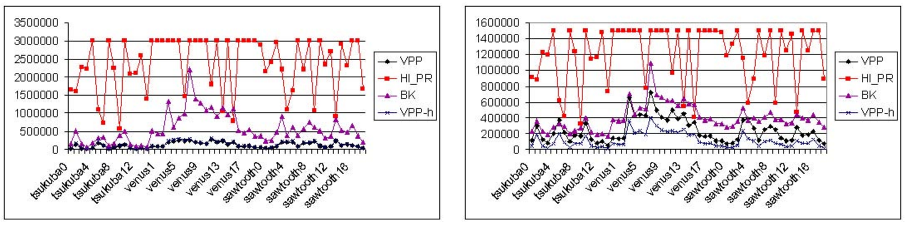


Fig. 3. Graph showing ratio of time for BK Vs VPP algorithms on BVZ test database

Figure 3 plots the ratio of time taken by BK and VPP. Note that BK is slower by at least a factor of 2 on 70% of the samples. Figures 4(a) and 4(b) compare the total number of nodes touched during push flow (expansion/augmentation) and relabelling(adopt orphans) phases in VPP, BK, and HI_PR. In both figures values have been truncated at the upper end. Note that the number of nodes touched during push flow phase in VPP are significantly smaller than those touched in BK and HI_PR. Poor performance of HI_PR is primarily due to use of approximate distance labels and repeated relabelling steps. In BK nodes touched in push flow is large as there is little control over augmenting path lengths and the amount of flow pushed in each path. However, nodes touched in the relabelling phase in BK is comparable to VPP.

It must be pointed out that total relabelling effort in BK is spread over a very large number of flow augmentation iterations. This effort is very small per flow augmentation iteration. This is because source and sink trees maintained by BK undergo very little change per iteration and effort involved (identification of disconnected nodes called orphans and rebuilding of trees by a process called adoption) is limited to searching in a small local neighbourhood in the grid graph. Also, since augmentation is not required to be on shortest paths global nature of the relabelling step in shortest path/distance label based algorithms is avoided. Note, however, that HI_PR which has local and global relabelling performs particularly poorly. This is not only due to the wasteful local relabelling steps but also because global relabelling cannot be made incremental as there is



(a) Nodes touched during push flow (b) Nodes touched during relabel

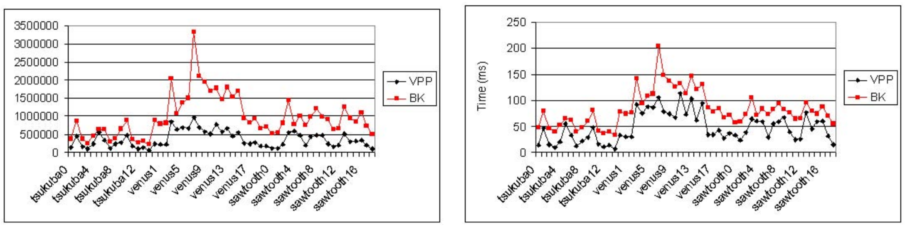
Fig. 4. Nodes touched

no obvious way to identify nodes whose distance labels will not change between two global relabelling steps in the presence of local relabelling.

We would like to point out that there is further scope for controlling relabelling in VPP. The current incremental relabelling strategy will assign non zero labels to sink cluster nodes at the cluster boundary that were neutralized ($excess(v) = 0$) during a push phase. The impact of this is to increase the number of nodes requiring relabelling in the Voronoi region graph though structurally graph may not have changed much. By carrying out partial labelling first from the shrunk boundary of a sink cluster to the original boundary we can determine the nodes of the original boundary which can still pass flow to the sink cluster nucleus. If we do this, we can effectively retain much of the original boundary of the sink cluster for the purposes of incremental relabelling. We call this the Hybrid VPP algorithm (VPP-h). Note that (Figures 4(a) and 4(b)) while nodes touched in the push phase in both VPP and VPP-h are similar number of nodes touched during relabelling in VPP-h is smaller. This is another instance where focus on the Voronoi subdivision of the grid graph has resulted in a heuristic to control relabelling costs.

Figure 5(a) shows total nodes touched in BK and VPP (sum of numbers in Figures 4(a) and 4(b)). It is interesting to see that time comparison in Figure 5(b) follows the trend showed by touched nodes graph in Figure 5(a). This is intuitive and logical since all the work in max-flow algorithms is concentrated in push and relabel operations.

Figure 6 has cumulative time, flow pushed and the number of nodes touched plotted as function of iteration number for one sample run of VPP algorithm. Note that all



(a) Total nodes touched (b) Time taken

Fig. 5. Comparing trends of total nodes touched and time taken in VPP and BK

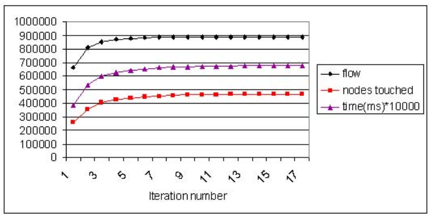


Fig. 6. Per iteration analysis for VPP for sample Sawtooth 9

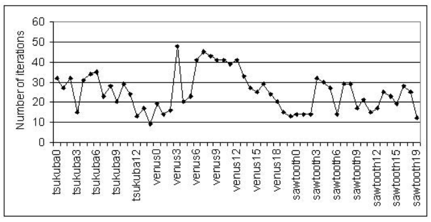
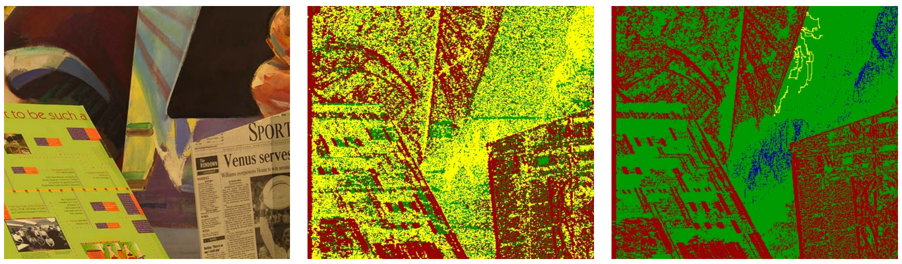


Fig. 7. Number of iterations on BVZ samples for VPP

the three curves have the same trend. Most of the time is taken as well as most of the flow gets pushed in the first iteration. As iteration number increases both the time taken and amount of flow pushed decrease as do the number of nodes touched. The implication is that Voronoi regions created become progressively smaller as iteration number increases. Also, useful work done per iteration is large in that ratio of the nodes relabelled and nodes involved in pushes in any iteration is high.

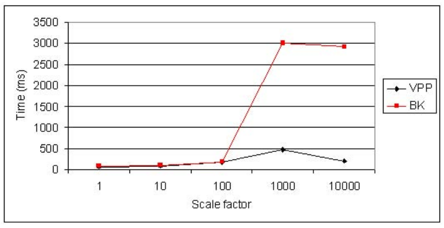
The above point gets made even more emphatically in Figure 8(b) and Figure 8(c) which show nodes involved (colored yellow) in a push operation during 1st and 35th iteration respectively when VPP is run on Venus7. It seems as if most of the nodes in the Voronoi region graphs around the sinks were involved in pushing flow from sources to the sinks. Since most of the pushes are saturated, one would expect that flow that reaches the sinks is large. This is indeed so. We have observed that about 90% of flow reaches the sinks in the first few iterations.

The worst case running bound of our algorithm is strongly polynomial compared to $O(n^3C)$ in case of BK. This is not simply an asymptotic curiosity. Relatively simple tweaks in link capacities can change the edges in such a way that BK slows down by as much as ten times. For the purpose of our experiments we took a sample (Sawtooth 9) from the dataset and scaled the capacities of n-links. Figure 9(a) shows the change in time. The reason for this time degradation is that the number of nodes touched during push flow starts to increase. Figure 9(b) shows the corresponding change in the number of nodes touched in the two algorithms during flow augmentation iterations.

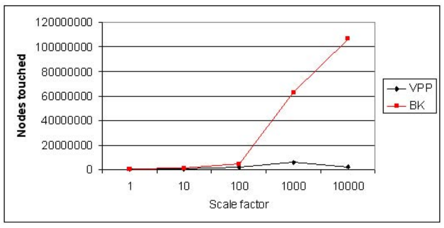


(a) Venus sample image (b) 1st iteration used nodes (c) 35th iteration used nodes

Fig. 8. Used nodes (shown in yellow) in one iteration



(a) Running time comparison



(b) Touched nodes comparison

Fig. 9. VPP and BK Comparison after scaling n-links in Sawtooth 9

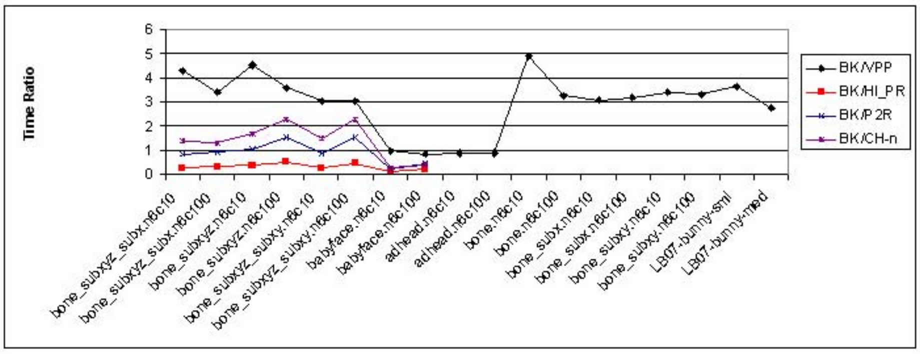


Fig. 10. Performance comparison of BK with VPP, HI_PR, PAR, P2R and CH-n on 3D, 6-connected datasets

Figure 7 shows the number of iterations taken by the VPP algorithm on BVZ samples. One interesting observation is that the upper bound on the number of iterations ($O(n)$) that one can formally prove overestimates the actual observed number significantly.

We have also compared performance of VPP and BK by running the two algorithms on 6 connected 3 dimensional data sets consisting of bone, babyface, bunny and adhead samples [3]. Figure 10 plots the ratio of time taken by BK and VPP, HI_PR, P2R, and CH-n. Ratio of time taken by BK and HI_PR, P2R, and CH-n are as reported in Table 5 in [19]. Note that VPP's performance is comparable to BK's on those samples for which Goldberg's set of algorithms (HI_PR, P2R, CH-n) are slower than BK [19]. On those samples (bone) for which CH-n, P2R are slightly faster than BK, VPP is 3 to 5 times faster.

4 Conclusions

The VPP algorithm presented above uses multiple paths with a single labelling, collects all flow first at a node before pushing, and partitions the grid flow graph in Voronoi regions at each iteration. Flow maintained after each iteration is a preflow but unlike the traditional preflow push algorithms there are two distinct phases in an iteration. The relabelling phase rebuilds the Voronoi region graphs, and push flow phase uses highest label first push and then breadth first inside sink cluster to push flow in the preflow framework in each Voronoi region. As sinks get saturated their Voronoi regions is redistributed among Voronoi regions which are still active. The algorithm improves upon the earlier reported algorithms both in terms of performance over standard data sets as well as demonstrably strongly polynomial worst case bound. This is important as we show cases where performance of algorithms without this property degrades quickly. We would like to re-iterate that our set of algorithms attempts to control the way a sink cluster can hold on to its ”Voronoi region”. There could be number of other strategies, around this central theme of ”Voronoi based preflow push” to improve the performance of graph cut algorithms for vision flow problems.

It should be noted that flow pushing is a symmetric activity, i.e. the end graph cut and max-flow obtained is independent of whether the flow is being pushed from so called sources to sinks or from sinks to sources. All that would happen is that the initial Voronoi regions created would change both in numbers as well as in shape. Actual time taken would also change. While we provide no formal proof, time taken would be proportional to the number of initial Voronoi regions and this can be used to improve performance. We have shown that relabelling costs are controllable by exploiting properties of the vision problems. Hybrid VPP is an example. In higher dimensions the grid structure becomes even more important as the number of edges in the grid graph start dominating. One way to use the higher dimensional grids effectively will be to model the neighbourhood locality structure more precisely. That is, distinguish between edges which are incident on nodes which are grid neighbours and those which are further apart. Preliminary experiments have suggested that in higher dimensions such locality impacts performance of algorithms compared here. How does it do so in higher dimensions is a theme we are exploring.

The authors would like to thank Niloy Mitra and the referees for their comments, inputs and careful reading of the manuscript.

References

1. Boykov, Y., Kolmogorov, V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. IEEE Trans. Pattern Anal. Mach. Intell. 26, 1124–1137 (2004)
2. http://vision.middlebury.edu/stereo/code/
3. http://vision.csd.uwo.ca/maxflow-data/
4. Boykov, Y., Funka-Lea, G.: Graph cuts and efficient n-d image segmentation. Int. J. Comput. Vision 70, 109–131 (2006)
5. Kolmogorov, V., Zabih, R.: Computing visual correspondence with occlusions using graph cuts. In: Proceedings of the International Conference on Computer Vision, vol. 2, pp. 508–515 (2001)
6. Kwatra, V., Schödl, A., Essa, I., Turk, G., Bobick, A.: Graphcut textures: Image and video synthesis using graph cuts. ACM Transactions on Graphics, SIGGRAPH 22, 277–286 (2003)
7. Lempitsky, V., Boykov, Y., Ivanov, D.: Oriented visibility for multiview reconstruction. In: European Conference on Computer Vision, vol. 3, pp. 226–238 (2006)

8. Kolmogorov, V., Zabih, R.: What energy functions can be minimized via graph cuts? IEEE Transactions on Pattern Analysis and Machine Intelligence 26, 147–159 (2004)
9. Greig, D.M., Porteous, B.T., Seheult, A.H.: Exact maximum a posteriori estimation for binary images. J. R. Statist. Soc. 51, 271–279 (1989)
10. Komodakis, N., Tziritas, G., Paragios, N.: Performance vs computational efficiency for optimizing single and dynamic mrfs: Setting the state of the art with primal-dual strategies. Comput. Vis. Image Underst. 112, 14–29 (2008)
11. Delong, A., Boykov, Y.: A scalable graph-cut algorithm for n-d grids. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–8 (2008)
12. Juan, O., Boykov, Y.: Active graph cuts. In: IEEE Conference on Computer Vision and Pattern Recognition (2006)
13. Juan, O., Boykov, Y.: Capacity scaling for graph cuts in vision. In: Proceedings of the International Conference on Computer Vision, pp. 1–8 (2007)
14. Boykov, Y., Veksler, O., Zabih, R.: Fast approximate energy minimization via graph cuts. IEEE Transactions on Pattern Analysis and Machine Intelligence 23, 1222–1239 (2001)
15. Ishikawa, H.: Exact optimization for markov random fields with convex priors. IEEE Transactions on Pattern Analysis and Machine Intelligence 25, 1333–1336 (2003)
16. Dinic, E.A.: Algorithm for solution of a problem of maximum flow in networks with power estimation. Soviet Math. Dokl. 11, 1277–1280 (1970)
17. Goldberg, A.V., Tarjan, R.E.: A new approach to the maximum-flow problem. Journal of the Association for Computing Machinery 35, 921–940 (1988)
18. Cherkassky, B.V., Goldberg, A.V.: On implementing push-relabel method for the maximum flow problem. Algorithmica 19, 390–410 (1997)
19. Goldberg, A.V.: Two-level push-relabel algorithm for the maximum flow problem. In: Goldberg, A.V., Zhou, Y. (eds.) AAIM 2009. LNCS, vol. 5564, pp. 212–225. Springer, Heidelberg (2009)
20. Vineet, V., Narayanan, P.J.: Cuda cuts: Fast graph cuts on the gpu. In: Computer Vision and Pattern Recognition Workshop, pp. 1–8 (2008)
21. `http://www.cs.ucl.ac.uk/staff/V.Kolmogorov/software.html`
22. `http://www.igsystems.com/hipr/index.html`
23. Chandran, B.G., Hochbaum, D.S.: A computational study of the pseudoflow and push-relabel algorithms for the maximum flow problem. Oper. Res. 57, 358–376 (2009)