Active Graph Cuts

Olivier Juan
CERTIS, Ecole Nationale des Ponts et Chaussees
Champs-sur-Marne, France
juan@certis.enpc.fr

Yuri Boykov
Computer Science Dpt., University of Western Ontario
London, ON, Canada
yuri@csd.uwo.ca

Abstract

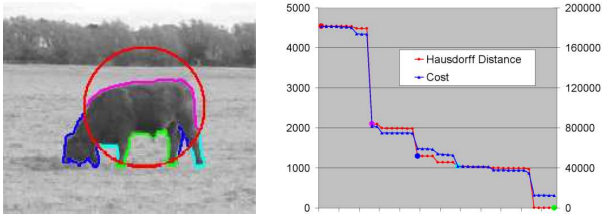
This paper adds a number of novel concepts into global s/t cut methods improving their efficiency and making them relevant for a wider class of applications in vision where algorithms should ideally run in real-time. Our new Active Cuts (AC) method can effectively use a good approximate solution (initial cut) that is often available in dynamic, hierarchical, and multi-label optimization problems in vision. In many problems AC works faster than the state-of-the-art max-flow methods [2] even if initial cut is far from the optimal one. Moreover, empirical speed improves several folds when initial cut is spatially close to the optima. Before converging to a global minima, Active Cuts outputs a multitude of intermediate solutions (intermediate cuts) that, for example, can be used be accelerate iterative learning-based methods or to improve visual perception of graph cuts real-time performance when large volumetric data is segmented. Finally, it can also be combined with many previous methods for accelerating graph cuts.

1. Introduction and Related Work

Our *Active Cuts* method has three major properties which, to the best of our knowledge, are fairly unique for s/t cuts algorithms.

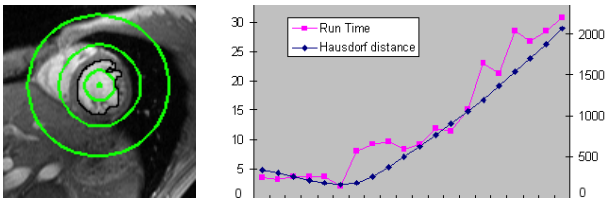
- *Initial Cut*: Normally, min-cut/max-flow algorithms compute global optima solutions which do not depend on any initialization. This explains why standard combinatorial techniques do not really use a concept of initial cut. However, applications in vision often give some reasonable initial guess. Active Cuts is a new approach to solving max-flow/min-cut problems that can start from any initial cut. Its running time directly correlates with a Hausdorff distance between initial cut and a globally optimal cut it computes (Fig.2).

- *Intermediate Cuts*: Standard combinatorial optimization algorithms generate only one good solution (global optima cut) computed at termination. In contrast, our method outputs a sequence of cuts of decreas-



(a) Active cuts (b) Cost of intermediate cuts

Figure 1. Active cuts in image segmentation (a). Initial cut is shown in red color. Intermediate cuts (displayed in different colors) gradually "carve" out the global minima solution that accurately follows object boundary. The cost of intermediate cuts and their Hausdorff Distance to the global optima decrease in time (b).



(a) Initial cuts (b) Run time

Figure 2. Active cuts can start from any initial cut (green circles) in (a). Plots (b) show that the running time until convergence to a global minima strongly correlates with a Hausdorff distance between initial cut and the actual minimum cut. The horizontal axis in (b) shows the radius of initial solution.

ing cost (Fig.1) as it converges to a global minima. In computer vision, it is often beneficial to have a multitude of good solutions which could be used for learning and robustness analysis. Intermediate cuts can be efficiently used in dynamic applications. They can also improve interactivity and visual perception of the real-time performance of graph cuts on large data sets.

- *Symmetry*: Active Cuts approach adds symmetry into standard combinatorial optimization algorithms for s/t cuts [6, 7]. We use *pseudoflows* [7] allowing nodes with positive flow *excesses* and negative *deficits*. However, our algorithm takes a symmetric approach complementary to [7]. Both excesses and deficits are actively pushed/pulled in the opposite directions away from a given initial cut towards the terminals s and t.

Recently, significant research efforts were devoted to efficiency of max-flow/min-cut methods in image segmentation [2, 9, 10]. We tested AC method in some basic applications of graph cuts to object/background extraction [1]. Despite polynomial complexity of graph cuts, computing globally optimal solutions for large images or volumes in real-time is still a challenge. Narrow bands can be used to improve efficiency [10, 13] by sacrificing global optimality. In contrast, we demonstrate a number of different ways where unique properties of Active Cuts allow to achieve much better practical efficiency without loosing globally optimality.

Even in tests with poor initialization Active Cuts gave either comparable or better speed than the max-flow technique in [2] which is widely used in vision. If a good initial solution is provided then Active Cuts compute globally optimal cut 2-10 times faster than [2], while initial solution is of no help to previous s/t cut algorithms. Theoretical worst case complexity of Active Cuts is similar to [2].

We also demonstrate applications of Active Cuts to other problems where initial solution is naturally available: dynamic video segmentation, object tracking, and hierarchical segmentation of large images/volumes.

Earlier methods for accelerating graph cuts in video were reusing flows from previous frames [9]. They use [2] as a basic algorithm but "recycle" flow and other data structures from frame-to-frame. Instead of recycling flows, our AC algorithm can recycle cuts from previous frames. We show that AC running time is proportional to the amount of motion between two consecutive frames. In our preliminary tests AC gave comparable or even better speed-up ratio with respect to [2] than the ratios reported in [9]. More importantly, however, in addition to recycling cuts AC algorithm can also recycle flows and other internal data structures which will likely give even stronger speed-up. In other words, it is possible to combine flow and cut recycling.

We also tested Active Cuts for hierarchical image segmentation which is important for large data sets. Initial cut for Active Cuts at a fine scale can be obtained from an optimal solution at a coarse scale. In contrast to earlier multi-level graph cut methods [10], we preserve global optimality.

The paper is organized as follows. We first present an overview of standard max-flow/min-cut algorithms (Section 2). Our Active Cuts method was inspired by a number of recent ideas and Section 3 explains how we combine them in a novel way based on our motivation to efficiently solve problems in computer vision. Sections 4 presents results of our experimental evaluation of Active Cuts on a number of generic problems in static, dynamic, and hierarchical image segmentation.

2. Standard Algorithms for s/t Graph Cuts

Combinatorial optimization community has a long history of research on algorithms for min-cut/max-flow problems on directed weighted graphs $\mathcal{G} = \langle \mathcal{E}, \mathcal{V} \rangle$. In the early 60's Ford and Fulkerson [5] showed duality between these two problems. In particular, they showed that a maximum flow from a given source $s \in \mathcal{V}$ to a given sink $t \in \mathcal{V}$ "saturates" a minimum cost s/t cut on $\mathcal{G}$. This result gave a first polynomial method for computing globally minimum s/t cuts via the *shortest augmenting paths* max-flow algorithm [5]. Later, Dinic [4] established solid grounds for better polynomial complexity max-flow algorithms by reducing the maximum network flow problem into a sequence of *blocking flow* problems. Until the late 80's, the majority of efficient min-cut/max-flow techniques were based on the fundamental work of Dinic [4]. Later generations of practically much more efficient algorithms for max-flow/min-cut problem were based on ideas generalizing the basic concept of a graph *flow* [8]. Our Active Cuts algorithm is using *pseudoflows*, which is one of such generalizations recently introduced by Hochbaum [7].

2.1. From *Feasible Flow* to *Preflow* and *Pseudoflow*

Earlier graph cut algorithms [5, 4] worked with feasible flows. Feasibility of a network flow normally implies that a flow at each specific edge should not exceed its capacity and that the total amount of inflow in_p that enters each node p should be equal to the amount of outflow out_p from that node. Relaxing flow conservation constraint was suggested by Karzanov [8] who used *preflows* where any graph node p can have a positive flow excess $\Delta_p = in_p - out_p \geq 0$. In fact, many min-cut/max-flow algorithms that are currently considered the state-of-the-art in the combinatorial optimization community are based on generalized network flows relaxing conservation constraint.

For example, preflow is an important component of a very widely used *push-relabel* algorithm of Goldberg [6]. Typically, the algorithm starts with an initial preflow where all edges from source s are saturated generating positive

excesses at nodes adjacent to s. These excesses are then *pushed* towards sink t based on node labels approximating distances to the sink. Pushing excess $\epsilon > 0$ from node p to q changes residual capacities of the edge (p, q) and its reverse edge (q, p) as follows

$$\begin{array}{rcl} w_{(p,q)} & = & w_{(p,q)} - \epsilon \\ w_{(q,p)} & = & w_{(q,p)} + \epsilon \end{array} \quad (1)$$

The excess values at nodes p and q are updated as well

$$\begin{array}{rcl} \Delta_p & = & \Delta_p - \epsilon \\ \Delta_q & = & \Delta_q + \epsilon \end{array} \quad (2)$$

Note that edge capacities have to remain non-negative and no edge can transmit an excess exceeding its current residual capacity. As edges become saturated (zero residual capacity), eventually the remaining excesses cannot find a path to the sink. Such excesses have to return back to the source. The algorithm terminates when all excesses disappear; they either reach the sink or return back to the source. At this point flow conservation is restored and algorithm's preflow turns into a feasible flow. This flow is also guaranteed to be a maximum flow.

The concept of preflow on a graph was recently further generalized by Hochbaum [7]. She uses *pseudoflow* allowing nodes with either positive *excesses*, $\Delta_p \geq 0$, or negative *deficits*, $\Delta_p \leq 0$. Similarly to push-relabel, Hochbaum's algorithm starts by saturating all edges from the source and creating positive excesses at nodes adjacent to it. Moreover, all edges to the sink are saturated too and negative deficits are created at nodes adjacent to the sink. Then, the algorithm pushes excesses towards the deficits along non-saturated edges using paths efficiently computed using an elegant system of dynamic trees. The algorithm outputs a minimum cost cut when excesses become completely disconnected from deficits by saturated edges. Unlike push-relabel algorithm, Hochbaum's method may terminate without a feasible maximum flow which, however, could be determined at additional computational cost.

2.2. Strategies for Pushing Flow

Intuitively, all standard max-flow/min-cut algorithms are different techniques for pushing flow/excesses from the source to the sink. Most methods agree that flow or excesses should be pushed along short non-saturated $s \rightarrow t$ paths (see Sec. 2.3 on techniques for finding such paths). There is much more variation in approaching other basic issues: how much flow/excess to push and how far.

For example, max-flow algorithms based on augmenting paths [5, 4] find a non-saturated path from the source to the sink, determine its bottleneck capacity, and push the corresponding amount of flow (excess) all the way from s to t. This is a "conservative" pushing strategy; no flow is ever pushed unless it is known that it can reach the sink.

In contrast, preflow and pseudoflow algorithms [6, 7] use "optimistic" strategies where flow/excess is pushed towards the sink in a sequence of local node-to-node steps. If some excess gets stuck at some node due to a weak local edge, there is a chance it may find an alternative path towards the sink (see Sec. 2.3). This is not guaranteed, however, and eventually some excesses get stuck.

2.3. Finding Paths via Dynamic Trees

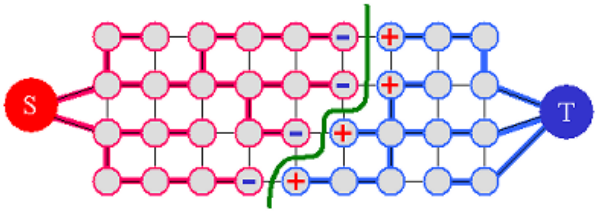
A technique for finding non-saturated paths between the source and the sink is an important component of all standard max-flow/min-cut algorithms. The main problem is that pushing flow/excesses along a path changes residual edge capacities and a set of non-saturated paths constantly changes. Efficient dynamic data structure for maintaining a system of non-saturated $s \rightarrow t$ paths is a must. Besides, such a system should maintain the shortest possible paths.

For example, the original augmenting paths algorithm of Ford and Fulkerson [5] recomputes the shortest augmenting $s \rightarrow t$ path after each iteration via Dijkstra's algorithm. However, pushing flow along a path may saturate only one edge and scanning the whole graph in order to recompute a new shortest path may not be the best approach.

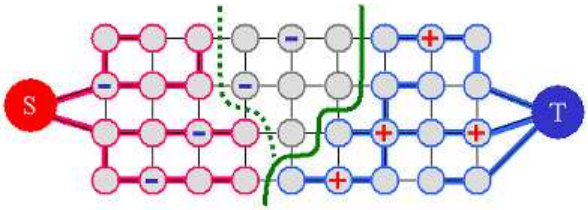
More efficient methods use *dynamic tree* data structures for path maintenance. Such trees are often initialized by Dijkstra's algorithm which may compute a tree of the shortest paths from the sink to other nodes. As an edge in that tree is saturated, the tree splits into two. Dynamic tree should efficiently add another non-saturated edge to restore connectedness while preserving short distances to the root, if possible. [12] proposed such dynamic trees in the context of augmenting path max-flow algorithms. [2] describe another dynamic tree structure that proved to be practical for large sparse grids common in computer vision.

Note that preflow and pseudoflow algorithms also use some explicit or implicit dynamic trees to determine what is a "good" path. For example, push-relabel algorithm [6] implicitly represents some tree (or a forest) via node labels. Rules for node relabeling implicitly update this tree/forest when some edges become saturated. Label-based approach to path-trees has one issue; when a node is relabelled its descendants are not automatically updated which is typical for explicit trees. Thus, paths in push-relabel may quickly become sub-optimal and, in practice, excesses motion pattern often resembles Brownian motion. This is why it is common to use *global relabeling* heuristic in push-relabel that once in a while runs Dijkstra's algorithm to restore a good labeling (tree).

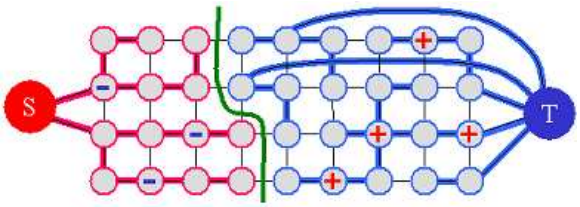
Hochbaum's pseudoflow-based algorithm [7] explicitly uses a forest of dynamic trees rooted at excess/deficit nodes. Each tree grows until some excess tree touches a deficit tree giving a non-saturated path from this excess to the corresponding deficit.



(a) Initialization from any given cut



(b) Some excess or deficit gets trapped



(c) Switching to a lower cost cut

Figure 3. The main steps of *Active Cuts* approach to solving minimum cost s/t cut problems on general directed graphs. We show a simple 2D grid example.

3. Active Cuts Method

Our approach to solving max-flow/min-cut problems is motivated by applications of graph cuts in computer vision and by new insights about existing methods in Section 2. Our method combines ideas of [6, 7, 2, 12] in a novel way. As discussed in Section 1, our approach has three fairly unique new aspects: arbitrary initial cuts, converging intermediate cuts, and symmetry.

3.1. Algorithm's outline

We are interested in a min-cut algorithm that can start at any given cut $C = \{\mathcal{S}, \mathcal{T}\}$ where cut C is defined by a binary partitioning of graph nodes into two subsets $\mathcal{S}$ and $\mathcal{T}$ such that $s \in \mathcal{S}$ and $t \in \mathcal{T}$. We will refer to subsets $\mathcal{S}$ and $\mathcal{T}$ as a source and a sink components of cut C.

One natural choice of initialization is a pseudoflow shown in Figure 3 (a). We can start from any initial cut C by saturating all edges on the boundary between its source and sink components $(\mathcal{S}, \mathcal{T}) = \{(p, q) \in \mathcal{E} : p \in \mathcal{S}, q \in \mathcal{T}\}$. This local edge saturation operation creates positive flow excesses $\Delta_q = w_{(p,q)}$ at boundary nodes $q \in \mathcal{T}$ on the sink side of the cut and negative flow deficits $\Delta_p = -w_{(p,q)}$ at the boundary nodes $p \in \mathcal{S}$ on the source side.

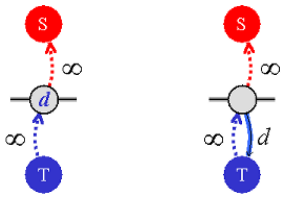


Figure 4. A deficit (a) is equivalent to a t-link to the sink (b). Indeed, pushing flow d along the t-link in (b) would saturate this t-link and create a deficit d as in (a). Similarly, any excess is equivalent to a t-link from the source. These ideas are described in [7].

In order to solve max-flow problem our method converts a pseudoflow into a feasible flow while maintaining some saturated cut. The general structure of Active Cuts method is summarized by the following one-line pseudocode:

ACTIVE CUTS:

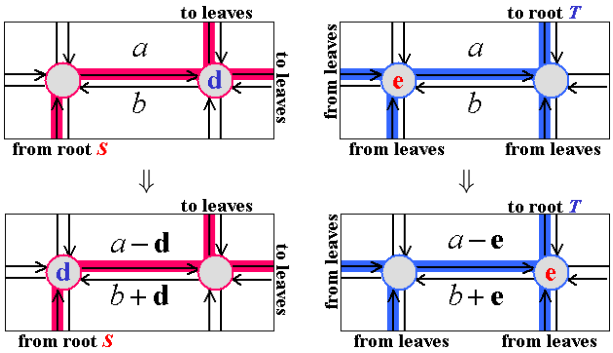
```
while (push_pull()==STUCK) recut();
```

Current cut C effectively separates the excesses and deficits into two sub-graphs $\mathcal{S}$ and $\mathcal{T}$ (Fig.3). Operation `push_pull()` should independently "push" excesses (+) towards the sink and "pull" deficits (-) towards the source keeping them separated within $\mathcal{T}$ and $\mathcal{S}$, correspondingly. On each sub-graph we can use any of the flow pushing strategies discusses in Section 2.3. Our only requirement is that function `push_pull()` should return in polynomial time with two possible outcomes:

- `STUCK`: some excesses or deficits are separated from the corresponding terminal by saturated edges, Fig.3b
- `DONE`: all excesses and deficits reached the terminals

In particular, `push_pull` can be implemented by directly applying push-relabel style operations [6] independently in each sub-graph. Section 3.2 describes practically more efficient implementation of operation `push_pull` using two independent dynamic trees similar to [2]. These trees have roots at the source and at the sink (see Fig.3) and span nodes using non-saturated edges.

As easy to show, if `push_pull` returns `STUCK` then we found a lower cost cut, Fig.3(b). In fact, the new cut is cheaper than the previous one by the total amount of trapped excess (or deficit). The `recut` operation should simply eliminates all trapped excesses or deficits by converting them into t-links as shown in Figure 4. After recutting, Fig.3(c), we can continue to `push_pull` excesses and deficits towards the terminals. The algorithm stops with a globally optimal cut when `push_pull` returns `DONE`.



(a) Pulling *deficit* to source (b) Pushing *excess* to sink

Figure 5. Excesses and deficits move towards the terminals on a directed graph using two dynamic trees with source and sink roots.

3.2. Implementing PUSH_PULL via dynamic trees

As illustrated in Figure 3, we use two dynamic trees; a *source tree* (red) directs deficits to the source and a *sink tree* (blue) directs excesses to the sink. Figure 5 zooms in some details in each tree. Note that the source tree (a) connects nodes to the source via edges (red) that are non-saturated in the direction from the terminal towards the leaf nodes. The sink tree (b) connects nodes to the sink via edges (blue) that are non-saturated in the direction from leafs towards the terminal. Due to saturated edges on the current cut C, the source and the sink trees can never overlap. As excesses and deficits move along the edges on the trees, we maintain short paths on both trees dynamically mainly based on ideas described in [2]. Both trees maintain the following properties:

- A tree is rooted at a terminal
- A tree spans all nodes "reachable" from a root
- All edges included in a tree are non-saturated

Note symmetric differences between the push and the pull operations illustrated in Figure 5. We use a push operation as in [6, 7]. A flow excess can be pushed from p to q only along non-saturated edge (p, q). The amount of sent excess should not exceed capacity $w_{(p,q)}$. Sending excess $0 < \epsilon \leq \Delta_p$ from p to q requires updates of edge capacities (1) and node excess values (2).

Unlike [6] and [7], we also actively move deficits using a symmetrically defined "pull" operation. A flow deficit can be pulled from node p to q only if reverse edge (q, p) is non-saturated. The amount of deficit sent should not exceed capacity $w_{(q,p)}$. In fact, sending deficit $0 > \epsilon \geq \Delta_p$ from p to q corresponds to the same updates (1) and (2) except that now ϵ is negative.

There a number of options in implementing the actual pushing/pulling. We chose to push the farthest excesses/deficits. We return STUCK as soon as some nodes can not be reached by a tree. This strategy generates a larger number of gradely moving intermediate cuts. These and some other tunings may affect practical performance. We chose the same tuning in all our experiments.

4. Applications and Evaluation

We tested AC method in applications of graph cuts to object segmentation [1]. Figure 2 clearly shows that the Hausdorff distance between initial and optimal segmentation is strongly correlated to the running time of the algorithm. We therefore expect that a "closer" initial cut will yield greater performance. Our experiments show that an initial cut based on a Voronoï partition still gives good results. However, the closer the initial cut to the optimal cut, the faster the convergence. Besides static object segmentation, we test this idiom in dynamic applications by *recycling cuts* from previous frames and in hierarchical tests where initial cut is obtained from courser scales.

4.1. Static Segmentation

For image segmentation, one should provide an initial cut for our algorithm. This could be a Voronoï partitioning (Fig. 6), a circular area provided by the user (Fig. 1 and 7) or some other prior (shape, color, texture...). However, this initialization should be carefully chosen as the speed of our algorithm is correlated with the Hausdorff distance between initial and optimal segmentation as shown in Figure 2.

We remarked that successive cuts often appear to **carve** the initial segmentation towards the global minima. A sequence of better cuts approaching a global optima can be viewed as a sequence of good local solutions and *may* be used in third party algorithms for estimating confidence of the segmentation results. These cuts may also prove useful for updating statistical properties of regions on the fly in applications such as GrabCuts [11].

Our algorithm is typically twice faster than [2]. However on some "difficult" examples, the two algorithms are comparable. Note, that pushing and pulling on two sides $\mathcal{S}$ and $\mathcal{T}$ of the current cut can be easily run in parallel on dual processors which may further improve our method.

4.2. Dynamic Segmentation

Given that our algorithm can take advantage of a close initial cut, one natural application is segmentation of dynamic video images. Spatio-temporal consistency between consequtive video frames allows to use an optimal cut from the previous frame as a good initial cut for the current frame (*cut recycling*). Note that spatio-temporal consistency of

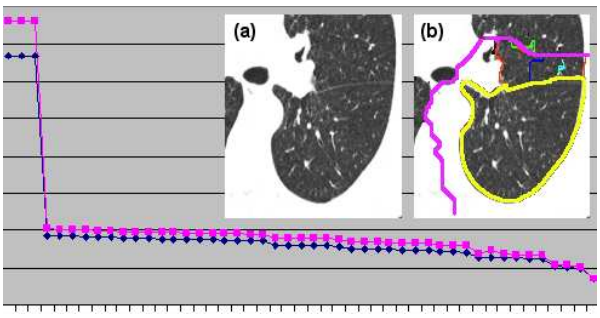
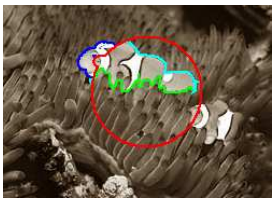
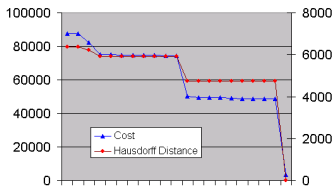


Figure 6. Lung lobe segmentation is difficult due significant image clutter (a) and a large number of strong local minima. Initial cut is shown in purple color (b). The final (global minima) cut accurately follows a faint fissure boundary between two lobes (in yellow). The cost of cuts (blue plot) and their Hausdorff distance (red plot) from the global optima decrease in time. Starting from a remote initial solution Active Cuts converges to a global minima twice as fast (33ms) as the state of the art max-flow algorithm in [2] (69ms).



(a) Active cuts



(b) Cost of intermediate cuts

Figure 7. Clown fish segmentation over structured background (a). Initial cut is a red circle given by the user. The final (global minima) cut accurately follows the fish external contours without becoming stuck on local minima in the clown disguise. The cost of cuts (blue plot) and their Hausdorff distance from the global optima (red plot) decrease in time (b). Active cuts converge to a globally minimal cut almost twice as fast (4.4ms) as the state of the art max-flow/min-cut algorithm in [2] (7.7ms).

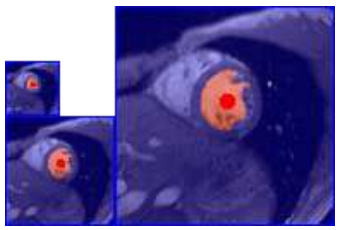
video data was recently used by Kohli and Torr [9] to accelerate graph cuts via *flow recycling* .

Figure 8 shows an example of video segmentation for hand tracking. On average, our algorithm is 5 times faster than the maxflow algorithm in [2]. Similar to conclusions from Figure 2, the time of convergence depends on the amount of change between two consecutive frames.

Note that the *flow recycling* technique [9] is fundamentally different from our *cut recycling* method. We believe that the two can be combined for a multiplied effect.

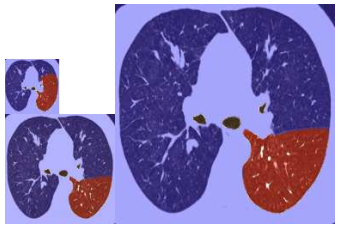
4.3. Hierarchical Segmentation

For static segmentation, ideally one should use prior knowledge to estimate the segmentation (shape, color, texture...). However, if no prior is available, a hierarchical approach provides an elegant way to initialize our algorithm by using segmentation of sub-sampled data. Such "coarse



Algorithm	Time
MaxFlow (BK)	18.15ms
ActiveCut	18.52ms
Hierarchical ActiveCut	Level 2 : 0.70ms Level 1 : 0.61ms Level 0 : 8.59ms Total : 9.90ms

Figure 9. Hierarchical segmentation using 16-Neighborhood. For each level, initial cut is set to an optimal cut/segmentation from the previous level, the deepest level is initialized using some image partition (like Voronoï partition). Speed of our algorithm is achieved when good initializations are provided. Hierarchical segmentation is an elegant way to estimate a minimum s/t-cut. Table gives timing comparison for maxflow/mincut algorithm in [2] and our method using standard initialization (Voronoï partition) and hierarchical initialization with only 2 levels of decimation. One can see a speed improvement of approximately a factor of 2.



Algorithm	Time
MaxFlow (BK)	26.47ms
ActiveCut	19.98ms
Hierarchical ActiveCut	Level 2 : 0.45ms Level 1 : 2.14ms Level 0 : 16.95ms Total : 19.54ms

Figure 10. Hierarchical segmentation using 4-Neighborhood and directed edges (dark object prior). Optimal segmentation from coarse scale is used to initialize fine scale. Speed of the hierarchical approach is better but deceiving in this case. However, the internal structure on the lung lobe makes it difficult to segment and the timing table confirms that ("*Level 0*") remains the most time-consuming computation.

cuts" have recently been used to accelerate graph cuts, but the narrow band technique of [10] sacrifices global optimality by limiting the scope of the solution. In contrast, we simply use the coarse cut as our initial cut on the full graph, thereby retaining global optimality. Figure 9 shows an example of hierarchical initialization on large neighborhood using only 2 levels of decimation. In this example, speed is improved by an order of 2 even though our hierarchical implementation itself is not optimized.

Our hierarchical algorithm does, on occasion, provide no significant speed up. Figure 10 shows a case were a Voronoï initial cut gives convergence as quickly as an initial cut generated by our hierarchical approach. By looking at the table, one can see that most time is spent in the "*Level 0*" step. The cluttered patterns inside the lung lobe makes segmentation more difficult because there are many strong local minima in the problem and the algorithm must explore all of them to find the global optima.

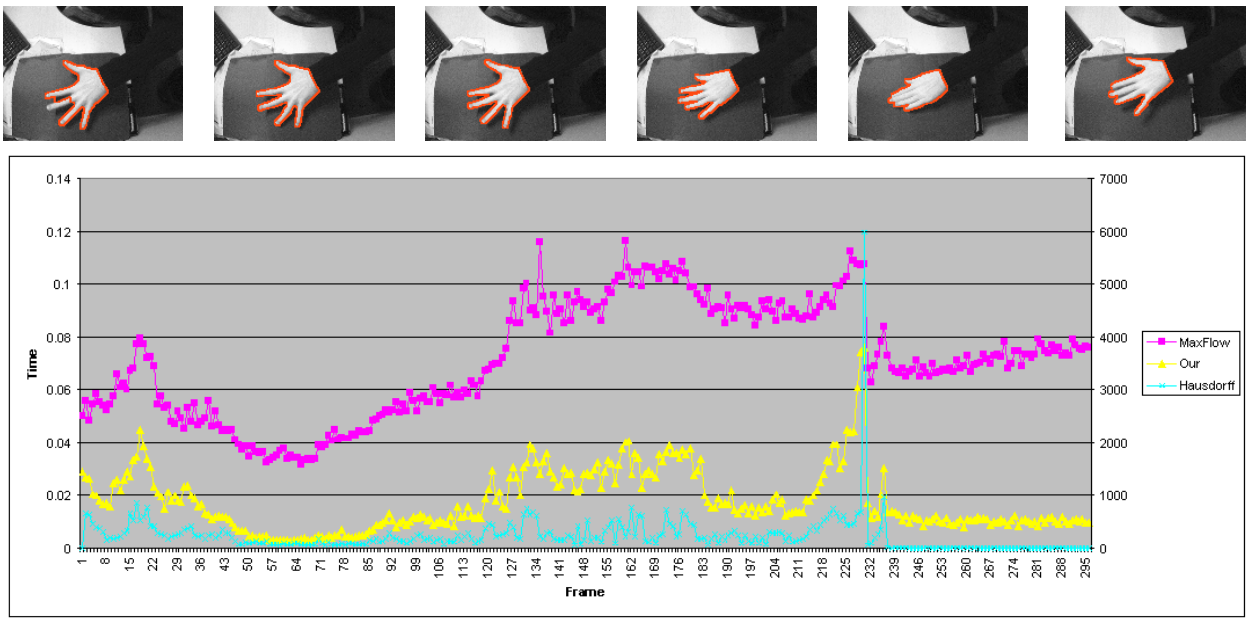


Figure 8. Dynamic segmentation of a video sequence. Active cuts algorithm (yellow) runs 2-6 times faster than the state-of-the-art max-flow algorithm in [2] (red). In each new frame initial cut is set from an optimal cut in the previous frame. Our algorithm's speed is almost linearly proportional to the magnitude of motion (Hausdorff distance between the segments in consecutive frames, in blue). Note that active cuts can be further accelerated in dynamic applications by "recycling" flow computed in the previous frame [9].

5. Conclusions and Future Work

Our new Active Cuts (AC) approach to max-flow/min-cut problems significantly improves practical efficiency (2-6 times) and applicability of graph cuts to many problems in image analysis. The method effectively uses initial cuts that are often available in dynamic and hierarchical applications. Such initial solutions are also available in iterative multilabel optimization techniques like α-expansions [3]. AC outputs a sequence of improving intermediate solutions that, for example, can be used be accelerate iterative learning-based methods, e.g. [11]. We are working on combining active cuts with *flow recycling* [9]. Narrow-band techniques [10, 13] can also benefit from our AC approach.

Acknowledgements

This work was supported by the NSERC (Canada) Discovery Grant R3584A02. We thank Vladimir Kolmogorov (University College London, UK) for pointing out important references. Our research has greatly benefited from the graph cuts visualization library of Andrew Delong (University of Western Ontario, Canada). We would also like to express our gratitude to Renaud Kerivien (Ecole Nationale des Ponts et Chaussees, France) and to Nikos Pargios (Ecole Centrale de Paris, France) for their strong support in organizing a collaboration between UWO and ENPC that lead to this research.

References

[1] Y. Boykov and G. Funka-Lea. Graph cuts and efficient N-D image segmentation. *International Journal of Computer Vision (IJCV)*, 70(2):109–131, 2006. Earlier in Y. Boykov and M.-P Jolly. *Interactive graph cuts* for optimal boundary & region segmentation of objects in N-D images. In *International Conference on Computer Vision*, vol. I, pp. 105-112, July 2001.

[2] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, September 2004.

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

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

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

[6] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *Journal of the Association for Computing Machinery*, 35(4):921–940, October 1988.

[7] D. S. Hochbaum. The pseudoflow algorithm for the maximum flow problem. *Manuscript, UC Berkeley*, revised 2003. Extended abstract in: The pseudoflow algorithm and the pseudoflow-based simplex for the maximum flow problem. Proceedings of IPCO98, June 1998. *Lecture Notes in Computer Science*, Bixby, Boyd and Rios-Mercado (Eds.) 1412, Springer, 325-337.

[8] A. V. Karzanov. Determinimg the maximum flow in a network by the method of preflows. *Soviet Math. Dokl.*, 15:434–437, 1974.

[9] P. Kohli and P. H. Torr. Efficiently solving dynamic markov random fields using graph cuts. In *International Conference on Computer Vision*, October 2005.

[10] H. Lombaert, Y. Sun, L. Grady, and C. Xu. A multilevel banded graph cuts method for fast image segmentation. In *International Conference on Computer Vision*, October 2005.

[11] C. Rother, V. Kolmogorov, and A. Blake. Grabcut - interactive foreground extraction using iterated graph cuts. In *ACM Transactions on Graphics (SIGGRAPH)*, August 2004.

[12] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.

[13] N. Xu, R. Bansal, and N. Ahuja. Object segmentation using graph cuts based active contours. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume II, pages 46–53, 2003.