Capacity Scaling for Graph Cuts in Vision

Olivier Juan and Yuri Boykov
Computer Science Department
University of Western Ontario, London, ON, Canada
juan@csd.uwo.ca, yuri@csd.uwo.ca

Abstract

Capacity scaling *is a hierarchical approach to graph representation that can improve theoretical complexity and practical efficiency of max-flow/min-cut algorithms. Introduced by Edmonds, Karp, and Dinic [7, 6] in 1972, capacity scaling is well known in the combinatorial optimization community. Surprisingly, this major performance improving technique is overlooked in computer vision where graph cut methods typically solve energy minimization problems on huge N-D grids and algorithms' efficiency is a widely studied issue [3, 12, 16, 10].*

Unlike some earlier hierarchical methods addressing efficiency of graph cuts in imaging, e.g. [16], capacity scaling preserves global optimality of the solution. This is the main motivation for our work studying capacity scaling in the context of vision. We show that capacity scaling significantly reduces non-polynomial theoretical time complexity of the max-flow algorithm in [3] to weakly polynomial $O(m^2n^2\log(U))$ where U is the largest edge weight. While [3] is the fastest method for many applications in vision, capacity scaling gives several folds speed-ups for problems with large number of local minima. The effect is particularly strong in 3D applications with denser neighborhoods.

1. Introduction

In computer vision max-flow/min-cut algorithms are largely seen as powerful global energy optimization techniques for N-D grids [5, 15, 13]. In this case *capacity scaling* for graph cut algorithms can be interpreted as a method for approximating energy functions at different accuracy levels. At the coarsest scale the energy function is only roughly approximated but it can be quickly optimized. Moreover, the coarse scale solution can be efficiently reused by max-flow/min-cut algorithms when moving to a better approximation of the energy at the next successive finer scale. At the finest scale one gets an exact global minimum of the original energy. Our goal is to demonstrate that computation of the global minimum for energies in computer vision can often be accelerated using the standard capacity scaling approach from combinatorial optimization. Interestingly, our tests on applications in image analysis show that even the coarsest scale solution is often very similar (if not identical) to a global minimum of the original energy.

The remaining part of the introduction outlines well-known connections between energy minimization and standard max-flow/min-cut algorithms. Section 2 reviews the standard capacity scaling framework for max-flow/min-cut algorithms. In Section 3 we discuss the implications of capacity scaling for optimization on N-D grids in the context of image analysis. In particular, we demonstrate an improvement of theoretical time complexity and practical efficiency for a widely used in vision max-flow algorithm for grids [3]. We show that capacity scaling allows to generate good approximate solutions with a known quality bound at a fraction of the time required to converge to the global minimum. We also demonstrate how capacity scaling significantly improves performance in complicated examples with a large number of local minima. Finally, detailed experimental evaluation of the algorithm in [3] with and without capacity scaling are presented in Section 3.4. In particular, we show that capacity scaling has the strongest effect on running time efficiency for grids of larger neighborhoods and for grids of higher dimensions (e.g. in 3D).

1.1. Energy minimization in vision and graph cuts

This paper focuses on minimization of binary energies of the simplest form

$$E(f) = \sum_{p\in\mathcal{P}} D_p(f_p) + \sum_{(p,q)\in\mathcal{N}} w_{pq}\cdot\delta(f_p = 1, f_q = 0) \quad (1)$$

where $\mathcal{P}$ is a set of sites (usually pixels) and $\mathcal{N}$ is a neighborhood system (a set of connected pairs of sites). Each site p should be assigned a label $f_p \in \{0, 1\}$. The mapping from sites to labels $f = \{f_p | p \in \mathcal{P}\}$ is called a configuration. The optimal configuration of labels is selected based on energy E combining data fit terms $D_p(\cdot)$ and non-negative discontinuity penalties $w_{pq} \geq 0$.

978-1-4244-1631-8/07/$25.00 ©2007 IEEE

Global minimization of binary energies as in (1) is probably the most common use for graph cuts in vision. In particular, many powerful object/background image segmentation methods [1, 2, 17] and volumetric multi-view surface reconstruction techniques [18, 4] use such energies. More generally, any submodular (2-clique) binary energy [15] can be converted to a form in (1). While exact global optimization via graph cuts is restricted to submodular energies, approximate optimization of non-submodular functions can be done via powerful QPBO methods [14] that first double the number of variables (sites) but then use binary graph cuts to minimize an energy like (1) on a twice bigger graph. Also, widely used multi-label energy minimization methods such as α-expansion or α/β-swap [5] use s/t graph cuts minimizing binary energies like (1) as a subroutine.

To keep it simple and intuitive, this paper discusses minimization of energy (1) in the context of binary image segmentation [1]. In this case, two possible values of labels $f_p \in \{1,0\}$ at each pixel p can be interpreted as "object" and "background". The data term gives two costs $D_p(0)$ and $D_p(1)$ describing individual preference of pixel p between two labels. Discontinuity penalty w_{pq} describe associativity between two neighboring pixel p and q. Typically, $D_p(0)$ and $D_p(1)$ are based on how well intensity I_p of pixel p fits given color models of an object and a background, while penalties w_{pq} are based on intensity difference $|I_p - I_q|$. Data term $D_p()$ can also impose hard constraints on certain pixels (seeds).

The connection between global minimization of binary energy (1) and s/t graph cut problems in combinatorial optimization is fairly intuitive [1]. Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ composed of a set of nodes (vertices) $\mathcal{V}$ and a set of weighted directed edges $\mathcal{E}$. Nodes can be associated with sites (pixels/voxels). $\mathcal{V}$ also contains two special terminal nodes: source s and sink t. The set of edges $\mathcal{E}$ is composed of two different types of link between the nodes. The *t-links* connect regular nodes (pixels) to the terminals, while *n-links* connect pairs of neighboring pixels $\{p,q\} \in \mathcal{N}$. Each (directed) n-link (p,q) is assigned a positive weight $w_{pq} \geq 0$. The t-links (s,p) and (p,t) connecting a pixel to the terminals are assigned costs $w_{sp} = D_p(0)$ and $w_{pt} = D_p(1)$. An s/t cut $\mathcal{C} = \{\mathcal{S}, \mathcal{T}\}$ is a binary partitioning of nodes into source $\mathcal{S}$ and sink $\mathcal{T}$ components such that $s \in \mathcal{S}$, $t \in \mathcal{T}$, $\mathcal{S} \cap \mathcal{T} = \emptyset$, and $\mathcal{S} \cup \mathcal{T} = \mathcal{V}$. Any cut $\mathcal{C} = \{\mathcal{S}, \mathcal{T}\}$ has a cost

$$|\mathcal{C}| = \sum_{p \in \mathcal{S}, q \in \mathcal{T}} w_{pq}$$

which is a sum of weights of severed edges. It is easy to check that $|\mathcal{C}| = E(f)$ when each cut $\mathcal{C} = \{\mathcal{S}, \mathcal{T}\}$ is associated with configuration f such that $f_p = 1$ for $p \in \mathcal{S}$ and $f_p = 0$ for $p \in \mathcal{T}$ defining a one-to-one correspondence between cuts and labelings.

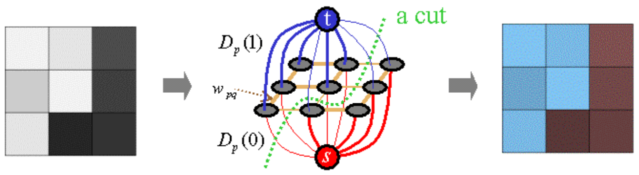

Figure 1. Minimization of binary energy (1) on images via graph cuts.

Thus, finding a globally optimal configuration f is equivalent to finding the minimum cost s/t cut on graph $\mathcal{G}$. It is well known that minimum s/t cut can be found via standard max-flow/min-cut algorithms from combinatorial optimization.

1.2. Overview of max-flow/min-cut algorithms

In 1955 Ford and Fulkerson, e.g. [8], showed that the problem of computing a minimum cost s/t cut on a graph is equivalent to fining a maximum s/t flow on the same graph. The max-flow problem can be intuitively formulated as follows. Each graph edge is seen as a pipe in a network and its weight corresponds to capacity to move water along that pipe. The source node is given the special role of flooding the graph with water, while the sink is the destination for this water. The max-flow problem can be formulated as: "*How much water could reach the sink from the source ?*"

Ford and Fulkerson showed that the maximum flow on the graph equals the cost of the minimum cut. The minimum cut can be seen as the bottleneck for the graph flow. Ford and Fulkerson also provided the first algorithm for solving the problem of maximum flow, the *augmenting path algorithm*. This algorithm iteratively finds non-saturated paths in the graph from the source to the sink and sends a flow along these paths until all paths have at least one saturated edge. Note that any non-saturated path from the source to the sink will do for the generic augmenting path approach of Ford and Fulkerson. Each augmentation increases the total flow between the terminals and the algorithms greedily converges to the maximum flow.

As was shown later, selection of paths significantly affects theoretical time complexity and practical efficiency of max-flow/min-cut algorithms. The generic method of Ford and Fulkerson has time complexity $O(m|C|)$ which is not even polynomial due to dependence on the cost of the minimum cut giving an upper bound on the number of augmentations. Edmonds, Karp, and Dinic proposed methods for path selection that reduce the number of required augmenting paths: *maximum capacity path* (fattest path) and *shortest path*. Various versions of these methods give max-flow/min-cut algorithms of strongly polynomial time complexity such as $O(m^2n)$ or $O(mn^2)$.

Ideally, one may want to combine the benefits of the shortest and the largest capacity paths; while we want a short path to reach the sink quickly, we also want to send as much flow as possible along each path. However, having both the shortest and the fattest path at the same time is practically impossible. Yet, a compromise could be obtained by relaxing the "maximum capacity" requirement and settling for a "sufficiently large capacity" instead. *Capacity scaling* approach proposed by Edmonds, Karp, and Dinic in 1972 suggest to selects the shortest path among all paths of capacity higher than a given threshold. Interestingly, capacity scaling generalizes to many max-flow/min-cut algorithms beyond augmenting path style technique of Ford and Fulkerson. For example, it can be applied to more recent methods based on pre-flows (e.g. *push-relabel* algorithm of Goldberg and Tarjan) and on pseudo-flows (Hochbaum [9]). Capacity scaling is widely used in the combinatorial optimization community to reduce theoretical complexity and to actual running time of graph algorithms. Section 2 provides a more detailed review of capacity scaling.

In the context of relatively sparse grids widely used in computer vision, an augmenting path algorithm [3] gives the state of the art empirical performance using dynamic trees. Yet, its worst case time complexity $O(mn^2|C|)$ is not polynomial. This complexity is worse than Dinic's or Edmonds and Karp's, but in practice [3] significantly outperforms them and other polynomial algorithms based on heuristics that work well on sparse grids. But, as [3] notes, empirical performance of the algorithm deteriorates on denser (larger neighborhood) grids and when moving from 2D to 3D applications. In fact, 3D grids are widely used in vision and larger neighborhoods are known to reduce geometric artifacts [2]. This paper addresses the limitations of the algorithm in [3] using capacity scaling framework; time complexity is reduced to weakly polynomial and we show that the running time on denser grids may reduce several folds.

2. Review of capacity scaling

Edmonds, Karp, and Dinic introduced capacity scaling in 1972 in order to solve some problems of the shortest augmenting path approach illustrated in Fig. 2. Capacity scaling first concentrates on "fatter" paths that can quickly increase the graph flow in a small number of augmentations even though these paths may not be the shortest. Yet, the method does not try to find the "fattest" (highest capacity) path which would be computationally expensive. Instead, it looks for paths of capacity larger than a given threshold Δ. Finding such paths is easy since they lie within a set of edges whose capacity exceeds Δ. For example, one can use breadth-first search to find the shortest path along edges (p, q) such that $w_{pq} \geq \Delta$.

Capacity scaling is a hierarchical (multi-scale) approach where initial coarser scales focus on higher capacity paths while later finer scales handle remaining lower capacity

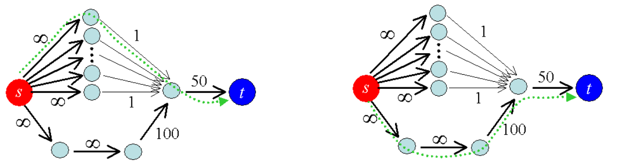

Figure 2. A number of required augmentations using the shortest paths (left) may be much larger then the number of augmentations based on "fatter" paths (right).

paths. At each given scale Δ, a max-flow/min-cut algorithm is run on a graph $\mathcal{G}_\Delta = \langle \mathcal{V}, \mathcal{E}_\Delta \rangle$ including the complete set of nodes and a subset of edges $\mathcal{E}_\Delta$ whose residual capacity is above the threshold. At scale Δ, the algorithm stops when there are no paths with capacity Δ or more.

When a coarser scale Δ_k is completed, the algorithm switches to a finer scale $\Delta_{k-1} < \Delta_k$ by adding graph edges with capacities $w_{pq} \geq \Delta_{k-1}$ exceeding the new threshold. The flow computed at a coarser scale remains feasible on a larger graph $\mathcal{G}_{\Delta_{k-1}}$ and graph cut algorithms can continue to increase that flow. At the new scale the algorithm can use paths of lower capacity Δ_{k-1}. Following the same procedure, one gets to the finest scale $\Delta = 0$ where all edges are present and the final max-flow (min-cut) solves the original graph cut problem exactly. Note that this approach avoids the problem outlined in Fig. 2 (left).

Selecting the right set of scales (thresholds) $S = \{\Delta_N, ..., \Delta_k, ..., \Delta_1, \Delta_0 = 0\}$ is important as it can change the complexity and the running time of graph cut algorithms. Assume that U denotes the highest edge capacity inside the graph. Obviously, no path on the graph has capacity higher than U and it would be useless to start at the coarsest scale $\Delta_N > U$. Consider two extreme sets of scales $S_\infty = \{U, U-1, ..., 2, 1, 0\}$ and $S_\emptyset = \{0\}$ in the context of augmenting path methods. In case of $S_\emptyset = \{0\}$ there are no constraints on the capacity of paths and capacity scaling reduces to the non-polynomial approach of Ford and Fulkerson. Choosing S_∞ leads to the highest capacity augmenting path algorithm which could be too expensive due to significant overhead. A balanced number of scales is often achieved by a geometrically decreasing series of thresholds like $S = \{2^{\log U}, ..., 2^k, ..., 2, 1, 0\}$ which can convert some non-polynomial max-flow algorithms to a weakly polynomial worst case complexity.

2.1. Improving time complexity of augmenting path algorithms

In this section, we outline how capacity scaling can change the complexity of the algorithms. Consider the set of scales $S = \{2^{\log U}, ..., 2^k, ..., 2, 1, 0\}$. Then, Lemma 2.1 gives an upper bound on the incremental flow that can be pushed at the lower/finer scales.

Lemma 2.1. *The maximum flow on graph $\mathcal{G}$ is bounded*

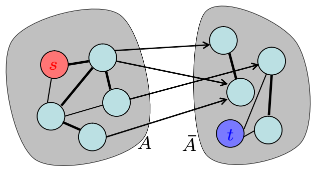

Figure 3. Illustration for lemma 2.1. At the end of scale $\Delta = 2^k$ any path from s to t has (bottleneck) capacity lower than Δ. Thus, the graph can be partitioned into two disjoint sets A and $\bar{A}$ such that all arcs from A to $\bar{A}$ have (residual) capacity lower than Δ. The thick and thin connections in the drawing above represent edges with (residual) capacity above and, correspondingly, lower than Δ.

above by $|f_k| + 2^k \cdot m$ where $|f_k|$ is the amount of flow that reached the sink at scale $\Delta = 2^k$.

This lemma implies that after scale $\Delta = 2^k$ the graph flow could increase in the remaining scales only at most by $2^k \cdot m$. The proof is based on the following observations: at the end of scale $\Delta = 2^k$ the graph nodes can be partitioned into two parts A_k and $\bar{A}_k$ where all arcs leaving A_k have residual capacities lower than 2^k while an upper-bound of the number of arcs between A_k and $\bar{A}_k$ is m (the number of edges). Note that a set of partitions A_k obtained at different scales define a sequence of coarse-to-fine approximations $C_k = \{A_k, \bar{A}_k\}$ of the minimum cut. Section 3.3 shows that such *approximate cuts* come with good quality bounds and that they may be useful in image analysis.

Lemma 2.2 provides an upper bound on the number of augmenting paths per scale.

Lemma 2.2. *The number of augmenting paths at scale k is at most $2m$.*

Proof. This is a direct consequence of the previous lemma. Each augmenting path at scale k has at least capacity 2^k. Since the remaining flow at the end of the previous scale $k+1$ is bounded by $2^{k+1}m$, the maximum number of augmenting paths is $2m$. □

Theorem 2.3. *The total number of augmenting paths is at most $O(m \log(U))$.*

Proof. Combine lemma 2.2 and the fact that the number of scales is $\log U$. □

In case of Ford and Fulkerson algorithm, the cost of a path is $O(m)$ and combined with capacity scaling the complexity of the algorithm becomes weakly polynomial $O(m^2 \log(U))$ instead of $O(m|C|)$ [7]. Later, Dinic [6] incorporated the shortest path strategy that further improved complexity to $O(mn \log(U))$.

3. Capacity scaling for problems in computer vision

In this section we describe a number of implications of applying capacity scaling to graph cut methods in computer vision. As explained in the introduction, we do not loose generality by concentrating on energy (1) and on binary image segmentation.

3.1. Integrating capacity scaling into the max-flow algorithm in [3] (BK)

As shown in [3], their version of augmenting path algorithm outperforms many standard max-flow/min-cut methods on sparsely connected grids common in computer vision. [3] is based on dynamic trees using a number of heuristics to efficiently maintain paths. Yet, their algorithm looses efficiency on denser graphs (with larger neighborhoods and in 3D). In part, this behavior can be explained by a non-polynomial worst-case complexity of this method $O(mn^2|C|)$ which explicitly depends on the cost of the optimal cut. For example, on "simpler" problems (e.g. sparse grids) the algorithms works significantly better than well known strongly polynomial methods like shortest paths (Dinic, Edmond, and Karp) or push-relabel (Goldberg and Tarjan). But, as was shown in [3], their algorithm's relative performance is miserable on denser test graphs commonly used for benchmarking in the combinatorial optimization community.

Relatively high worst-case complexity $O(mn^2|C|)$ of the algorithm in [3] comes with dependence on the cost of the optimal cut. Similarly to the algorithm of Ford and Fulkerson, $|C|$ is the only bound on the number of required augmentations available for [3]. Section 2.1 shows that capacity scaling can significantly improve that bound. The practical problems addressed by capacity scaling (see Section 2.1 and Fig. 2) are fairly representative of problems that occur on graphs in image segmentation and in other computer vision problems (see Section 3.2). The potential of improving theoretical complexity and practical running time performance provided us with substantial motivation to incorporate capacity scaling into the graph cut algorithm in [3].

By applying capacity scaling to Boykov-Kolmogorov (BK) algorithm [3], we are able to reduce its complexity from $O(mn^2|C|)$ to $O(m^2n^2 \log U)$. This complexity improvement follows from Theorem 2.3 and the fact that BK does at most $O(mn^2)$ operations per each augmenting path. Therefore, a combination of capacity scaling and dynamic trees in [3] results in a weakly polynomial max-flow algorithm (BKCS). As we show in Section 3.4, BKCS outperforms BK mostly on denser grids (high dimension, big neighborhood) and it is comparable to BK on simpler problems.

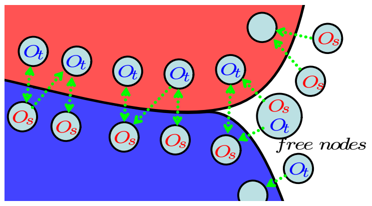

Figure 4. The red and blue regions correspond to the source and the sink trees respectively. The nodes outside of these regions are *free* nodes that were not reached by the trees due to the scale Δ constraint. The O_s and O_t are respectively the *outer* nodes - the first neighbors of the source and sink tree respectively. The green arrows show the "reactivation" relationship between the outer and inner nodes based on graph edges.

Note that BK [3] uses two dynamic trees (rooted at the source and the sink) that help to find augmenting paths. To implement BKCS we would like to preserve these trees between the scales as rebuilding them between two scales would be inefficient (similarly to *dynamic cuts* [12]). Moreover, preserving the trees from coarser scales reinforces a more balanced structure for the trees at the finer scales: stronger edges will be closer to the roots and weaker edges will be closer to the leaves. This happens because preserved branches from coarser scales will be stronger (fatter) than the new weaker branches added at a finer scale. Maintaining a strong balanced structure for the trees has a strong impact on practical performance since it helps to avoid unnecessary saturations closer to the root. Generally speaking, edge saturations closer to the root are harder to fix as they require rebuilding a big part of the tree.

Below we discuss some critical modifications that we had to implement for BKCS in order to achieve efficient integration of capacity scaling (CS) and dynamic tree structures in BK. Preserving the trees between two scales is not entirely straightforward. Like in Dijkstra's, BK maintains a list of *active nodes* located on the border of two trees in order to grow them. The algorithm terminates when the two trees cannot grow anymore. In this case there are no more *active nodes*. The same occurs at the end of each scale of BKCS. In order to reuse the trees for next scale, one needs to reactivate the nodes on the border of the trees and those nodes may not be the leaves (see Fig. 4).

To efficiently reactivate the nodes, we keep track of the nodes located on the outer border of the two trees using two (independent) lists and by flagging the nodes. Then, reactivating the inner border nodes becomes easy since they are all neighbors of the outer border nodes. This approach is easier and more efficient than trying to keep track of the inner border nodes directly. While our approach to reactivating nodes between two scales is relatively efficient, it still presents a non-negligible computational overhead of BKCS when compared to BK.

For completeness, we also implemented capacity scaling for push-ralabel algorithm. [3] shows that on dense 3D grid, Push-Relabel challenges the state of the art algorithm in vision. In the next section, we compare BKCS to Push-Relabel (PR) (with both Global Relabel and Gap Relabel heuristics) on some vision problems.

The Push-Relabel algorithm is a very different and elegant algorithm. It uses a labeling method to encode the structure of the graph instead of dynamic trees. A "labeling" is in fact an implicit representation of a tree in a graph. To each node is assigned a label which is a lower bound estimation of the distance to the sink. It is a different but yet "equivalent" representation of a graph structure since one can build a dynamic tree from a labeling by following the natural order relation between nodes induced by the labels (and vice-versa).

The modifications of Push-Relabel are pretty much natural and the overall algorithm is easy to implement. However, at the end of each scale, the labeling is no more valid for the next scale. Different approaches had been tried to restore the labeling efficiently. A naive way would be to trigger a global relabeling. This is equivalent to dropping the dynamic trees between two stages which has been proved inefficient (empirically). To efficiently restore the labeling, we used a similar technique as for BKCS but the overhead is still important. This is mainly due to the lack of symmetry of Push-Relabel which implies to relabel a big region of the graph. One can expect better performance using a symmetric approach such as active cuts [10].

3.2. Multiple local minima

Any max-flow/min-cut algorithm finds a global minimum solution even in difficult cases when there is a large number of local minima. Yet, each algorithm's running time may strongly depend on the amount of clutter in an image. For example, Fig. 5(a2) shows that BK [3] uses only 1020 augmenting paths to find a global minimum for a synthetic image segmentation problem in (a1) but it takes 5572 augmentations (b2) to find the same global optimum solution when other local minima are added (b1).

Improved theoretical complexity of capacity scaling approach may suggest that the number of required augmentations for BKCS algorithm should be less sensitive to specific instances of image data of the same size. Fig. 5(a3,b3) shows that the number of augmenting paths required by BKCS is less sensitive to clutter (other local minima) in the image. This can be explained as follows. In case of no clutter (a1) all paths are equally good and both algorithms use approximately the same number of paths. Yet, capacity scaling can identify stronger (fatter) paths in more complicated cases with large amount of clutter in the image (b1). Obviously, there are much fewer "fat" paths (b3) but using them max-flow algorithms can reach the maximum faster.

Similar observations can be made on real images. For

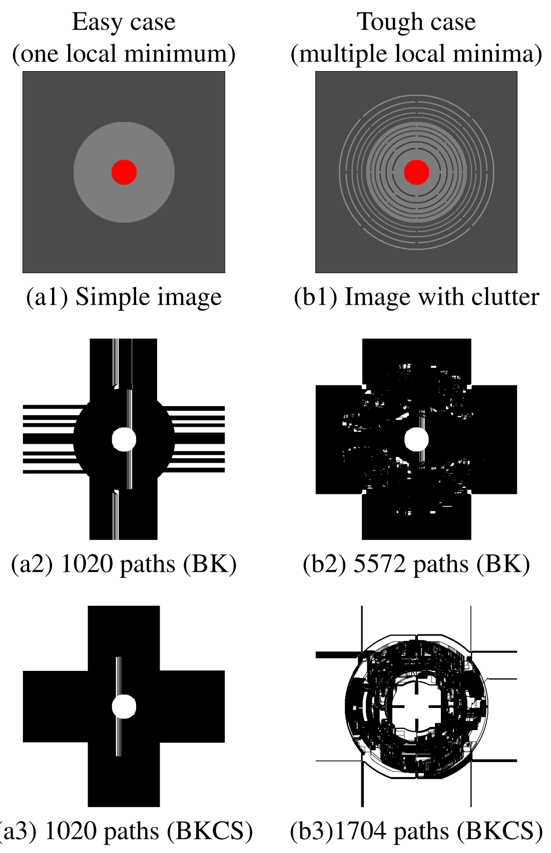

Figure 5. Augmenting paths generated by graph cut algorithms in image segmentation. We compare an image with one good solution (a1) and an image with multiple local minima (b1). The object seeds are red, the sink seeds are located on the image border. In both cases the global minimum is the same (a gray circle) and any max-flow/min-cut algorithm finds it. The second and third rows illustrate augmenting paths obtained with and without capacity scaling. Black color shows nodes that were used at least by one augmenting path.

example, a “flower” image in Table 3.4 has relatively small amount of clutter. Even then, BK needs 92711 augmenting paths while BKCS uses only 18532 (5 times less) paths to find the same minimum cut. There is even more clutter in the “lung” example in Fig. 7 where a global optima solution is a faint line (lobe fissure) surrounded by similar looking structures (vessels). In this case capacity scaling reduces the number of required augmenting paths almost 8 times (from 74258 for BK to 9881 for BKCS). Note that if there is absolutely no clutter, as in Fig. 5(a1), then BK and BKCS obtain the same number of augmenting paths.

Similar behavior can be seen when increasing noise. In that case, the noise generates more and more “local minima” when its level increases. Fig. 6 shows how the number augmenting path increases with the level of noise on a simple example (left image of Fig. 5).

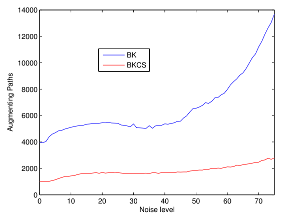

Figure 6. The number of augmenting path is dramatically increasing with the noise level for the standard BK algorithm, while for BKCS, this number stays stable. This will allow good performance on noisy and/or challenging data.

3.3. Fast approximate solutions with optimality bounds

As explained in Section 2, at the end of each scale $\Delta = 2^k$ capacity scaling approach generates an intermediate cut $C_k = \{A_k, \bar{A}_k\}$. In this section we will show that such intermediate cuts are good approximations of the global minimum cut. First of all, each cut C_k comes with a guaranteed optimality bound that typically gets tighter for finer scales. Note that the finest scale cut C_0 is guaranteed to be the exact global minimum. In practice, the actual deviation of the cost of C_k from the cost of the global minimum is very low even at the coarsest scales. Moreover, the image segmentation corresponding to C_k is often almost identical visually to the solution corresponding to the global minimum C_0. Fig. 7 demonstrates segmentation results corresponding to intermediate cuts C_k for two images.

First we will show the optimality bound that each intermediate cut C_k comes with. The results in Section 2.1 imply that

$$f_k \leq |C| \leq |C_k| \leq f_k + 2^k \cdot m$$

where $|C|$ is the cost of the global minimum cut and $|C_k|$ is the cost of the intermediate scale cut on a full graph G. Note that flow value f_k and cost $|C_k|$ are known at the end of computations at scale $\Delta = 2^k$. This gives a relative error bound $\frac{|C_k|-f_k}{|C_k|}$ for the cost of C_k with respect to the global minimum. Plots at the bottom of Fig. 8 show these approximation error bounds and the actual errors with respect to the global minimum for intermediate cuts on images in Fig. 7. We also show similar plots for “typical” 3D segmentation examples (on real medical data) in Fig. 9. The actual errors in our tests were much lower than the upper bound.

The good quality of approximate cuts generated by capacity scaling allows to use them instead of the global minimum in imaging applications where speed is a priority. These coarse scale solutions are obtained at a fraction of the time it takes to converge to the global minimum and in many cases these solutions are as good. In particular, cuts C_k are safe to use in low clutter examples. Also, a decision to use an intermediate cut may be based on available quality bound value.

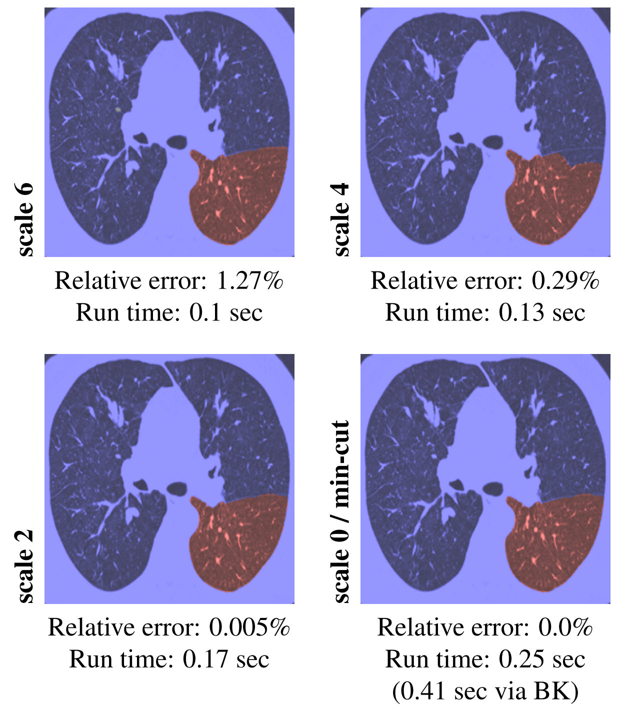

Figure 7. Each scale of *capacity scaling* returns an intermediate solution. These solutions are good approximations of the global minimum (bottom row) even though they are computed much faster. Note that the relative cost error (w.r.t. global minimum) is fairly small even at the coarsest scale (top row). Interestingly, intermediate solutions at the coarser scales are hard to distinguish visually from the global minimum (bottom row). However, in challenging cases with substantial clutter some scales may return a local minima (see "lung", scale 4).

3.4. Experimental comparison of Capacity Scaling

In the previous section we showed that capacity scaling can produce very good quality approximate cuts C_k well before it converges to the guaranteed global minimum at the finest scale. Yet, in some cases it is important to have a guaranteed global solution. Our experiments below demonstrate the running time of capacity scaling approach to complete convergence at the finest scale. We compare it in Table 3.4 against the running time of BK algorithm [3] which is the state of the art max-flow algorithm for applications on sparse grids in computer vision. We also compare it to push-relabel (PR) algorithm which is the state of the art max-flow algorithm on dense grids in combinatorial optimization (we use Global and Gap Relabel heuristics).

For the sake of completeness, we also applied Capacity Scaling to Push Relabel. It turned out to be a failure. The speed and the complexity of PR are related to the number of non-saturating pushes while CS avoids unnecessary saturation and increases the number of non-saturating pushes. A better approach for PR is a technique called "Excess Scaling" but if it improves the worst case complexity, it increases the computational time due to a big overhead in the implementation (for more details, refer to [11].

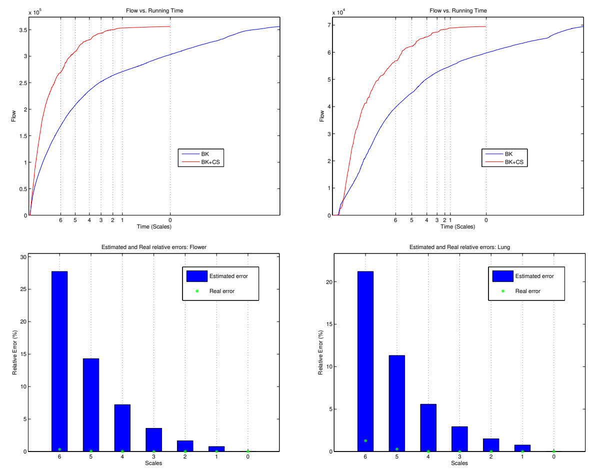

Figure 8. Capacity scaling for 2D images: "flower" (left) and "lung" (right), same images as in Fig. 7. First row: flow reaching the sink vs. running time for BK (blue) and BKCS (red). Vertical dotted lines correspond to the end of scales. Second row: estimated error bounds (blue) and real observed errors (green) of intermediate solutions produced after each scale of BKCS.

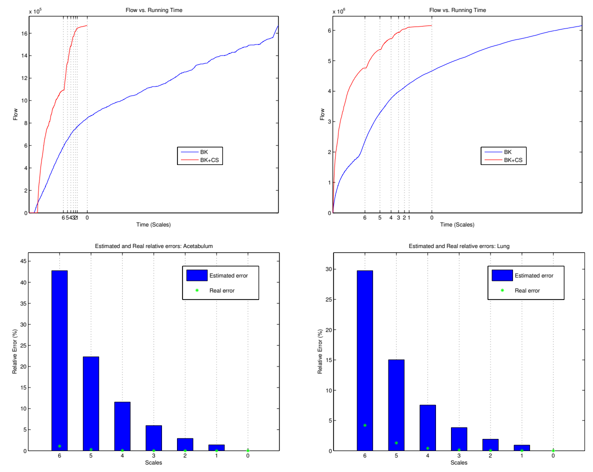

Figure 9. Capacity scaling for 3D image volumes: a hip bone (left) and a challenging lung lobe (right). See caption of Fig. 8 for details.

4. Conclusions and future work

Capacity scaling improves theoretical complexity of BK [3] to weakly polynomial by reducing the number of augmentations required to converge to the global minimum. BKCS outperforms BK on challenging graphs but the speed up can be deceiving on simple graph (e.g. 4-neighbor 2D) due to the overhead of capacity scaling. Yet, BKCS sig-

	Samples	Size	**BK** #Paths (Running Time)	**BKCS** #Paths (Running Time)	**Comparison BK/BKCS**	**PR** #Push (Running Time)	Comparison **PR/BKCS**
2D **16 Neighbors**	Ventricle	128x128	7554 (11ms)	1507 (6.9ms)	**1.6**	60155 (6.3ms)	**0.9**
	Fish	250x179	20177 (46ms)	3639 (24ms)	**1.9**	168681 (26ms)	**1.1**
	Coronary	256x256	57647 (136ms)	8517 (45.5ms)	**3**	736874 (137ms)	**3**
	Lung	256x256	74258 (282ms)	9881 (131ms)	**2.1**	671234 (116ms)	**0.9**
	Flower	400x300	92711 (296ms)	18532 (134ms)	**2.2**	1642615 (345ms)	**2.6**
	Neurovasc	512x512	229204 (786ms)	40482 (344ms)	**2.3**	2285073 (700ms)	**2**
3D **26 Neighbors**	Heart	128x128x86	297902 (1.4s)	78275 (1.3s)	**1.1**	3736132 (2.3s)	**1.8**
	Acetabulum	128x128x119	1066971 (15.8s)	162523 (2.7s)	**5.8**	11082897 (7.7s)	**2.9**
	Acetabulum	256x256x119	5610213 (185s)	647579 (29s)	**6.4**	69559385 (63s)	**2.2**
	Baby Face	250x25x81	19272649 (419s)	2865353 (123s)	**3.4**	162483987 (147s)	**1.2**
	Lung	205x165x253	18893807 (391s)	914817 (111s)	**3.5**	116791629 (122s)	**1.1**

Table 1. Timing for BK, BKCS, and PR on 2D and 3D segmentation on regular grid. CS reduces sensitively both the number of augmenting paths and the running time of BK algorithm. In [3], BK was outperformed by PR on dense 3D grid. Capacity Scaling fixes that since BKCS outperforms PR on dense graph. More experiments are available in the tech report [11].

nificantly outperforms BK and PR on denser grids and in cases with clutter (many local minima) and noise. The speed-up factor (up to 6) is particularly strong for 3D grids (volumes) and for larger neighborhoods (e.g. 26) which are important for reducing geometric artifacts of combinatorial graph cut methods in vision. Running time can be further improved (by a factor of 2-4) using approximate solutions that capacity scaling generates at coarser scales. These approximate solutions come with a certain optimality bound. Often they are visually indistinguishable from a global minimum solution. It is also possible to apply capacity scaling to active cuts [10] and to dynamic cuts [12]. An implementation will be available at `http://www.csd.uwo.ca/∼juan/software.html`.

References

[1] Y. Boykov and G. Funka-Lea. Graph cuts and efficient n-d image segmentation. *Int. J. Comput. Vision*, 70(2):109–131, 2006.

[2] Y. Boykov and V. Kolmogorov. Computing geodesics and minimal surfaces via graph cuts. In *IEEE International Conference on Computer Vision*, volume 1, pages 26–33, Washington, DC, USA, 2003.

[3] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(9):1124–1137, sept. 2004.

[4] Y. Boykov and V. Lempitsky. From photohulls to photoflux optimization. In *British Machine Vision Conference*, volume 3, pages 1149–1158, Edinburgh, UK, Sept. 2006.

[5] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(11):1222–1239, 2001.

[6] E. A. Dinic. Metod porazryadnogo sokrashcheniya nevyazok i transportnye zadachi. *Issledovaniya po Diskretnoi Matematike, Science*, 1973. Title translation: The Method of Scaling and Transportation Problems.

[7] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the Association for Computing Machinery*, 19(2):248–264, 1972.

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

[9] D. S. Hochbaum. The pseudoflow algorithm: A new algorithm for the maximum flow problem. Long version of same titled paper, may 2004.

[10] O. Juan and Y. Boykov. Active graph cuts. In *IEEE Conference on Computer Vision and Pattern Recognition*, june 2006. New York.

[11] O. Juan and Y. Boykov. Capacity scaling for graph cuts in vision. Technical report, University of Western Ontario, august 2007.

[12] P. Kohli and P. H. Torr. Efficiently solving dynamic markov random fields using graph cuts. In *IEEE International Conference on Computer Vision*, volume 2, pages 922–929, 2005.

[13] V. Kolmogorov and Y. Boykov. What metrics can be approximated by geo-cuts, or global optimization of length/area and flux. In *IEEE Inter. Conf. on Comp. Vision*, volume 1, pages 564–571, Washington, DC, USA, 2005.

[14] V. Kolmogorov and C. Rother. Minimizing non-submodular functions with graph cuts - a review. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2007. (to appear).

[15] V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(2):147–159, February 2004.

[16] H. Lombaert, Y. Sun, L. Grady, and C. Xu. A multilevel banded graph cuts method for fast image segmentation. In *IEEE International Conference on Computer Vision*, volume 1, pages 259–265, Washington, DC, USA, 2005.

[17] C. Rother, V. Kolmogorov, and A. Blake. Grabcut: interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.*, 23(3):309–314, 2004.

[18] G. Vogiatzis, P. H. S. Torr, and R. Cipolla. Multi-view stereo via volumetric graph-cuts. In *IEEE Comp. Soc. Conf. on Comp. Vision and Pattern Recog.*, volume 2, pages 391–398, Washington, DC, USA, 2005.