Efficient Planar Graph Cuts with Applications in Computer Vision

Frank R. Schmidt, Eno Töppe and Daniel Cremers
Computer Science Department
University of Bonn, Germany

Abstract

We present a fast graph cut algorithm for planar graphs. It is based on the graph theoretical work [2] and leads to an efficient method that we apply on shape matching and image segmentation. In contrast to currently used methods in Computer Vision, the presented approach provides an upper bound for its runtime behavior that is almost linear. In particular, we are able to match two different planar shapes of N points in $O(N^2 \log N)$ and segment a given image of N pixels in $O(N \log N)$. We present two experimental benchmark studies which demonstrate that the presented method is also in practice faster than previously proposed graph cut methods: On planar shape matching and image segmentation we observe a speed-up of an order of magnitude, depending on resolution.

1. Introduction

One of the major challenges for modern Computer Vision is the abundance of high-resolution image data. This poses an urgent need for algorithms with fast and predictable runtimes. Many of the computational challenges including image segmentation, stereo reconstruction or shape matching have recently been addressed by graph cut approaches, because these allow to efficiently solve the underlying labeling or correspondence problems in a globally optimal manner. In particular, researchers have employed graph cuts for stereo reconstruction with convex neighborhood potentials [4], for image segmentation [3, 10, 16], for multiview reconstruction [19, 21] or for planar shape matching [18].

To date the graph cut algorithm of Boykov and Kolmogorov is considered the fastest existing algorithm for these types of applications [4]. Nonetheless, there is no known polynomial upper bound for the runtime of the algorithm of Boykov and Kolmogorov. In fact, in applications of interactive segmentation its runtime typically depends on the initialization, *i.e.*, on the user-specified choice of pixels that belong to the object. A lot of effort has been put into improving the runtime of maximum flow computation,

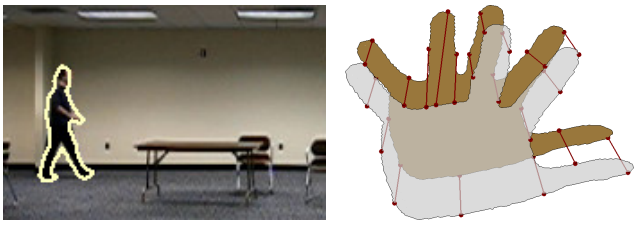

Figure 1. Planar graph cut applications. Numerous computational challenges like image segmentation or shape matching can be solved by means of planar graph cuts. In this paper, we propose an efficient algorithm for planar graph cuts which has better computational complexity, substantially higher speed and more predictable runtimes than currently used algorithms.

by means of flow recycling [13], capacity scaling [12] or multi-scaling [6]. While these strategies often lead to reduced computation times, none of them reduces the worst case complexity of the methods that they were built on. A recent comparison of graph cuts versus dynamic programming on the problem of planar shape matching [18] demonstrates the lack of a strong upper bound on the graph cut computation time: While the matching of *similar* shapes via graph cuts was faster than the commonly used dynamic programming approach, the matching of *very dissimilar* shapes was substantially slower. This absence of an upper bound can be an important restriction for the use of graph cuts in time-critical applications where each component of a complex system needs to provide its output within a specified time frame.

The goal of this work is to overcome this restriction for a certain subclass of graph cut applications, namely for planar graphs. Such planar graph cuts include the cases of planar shape matching [18] and that of image segmentation approaches such as the intelligent scissor method [15]. Firstly, we propose a planar graph cut algorithm which is built on the method of Borradaile [2]. Secondly, we will show in two benchmark tests on planar shape matching and image segmentation that the proposed algorithm outperforms existing graph cut solutions both in terms of worst-case complexity and in terms of actual computation times.

This work is organized as follows. In Section 2, we present some notations and revisit ideas of graph cut meth-

978-1-4244-3991-1/09/$25.00 ©2009 IEEE

ods. In Section 3, we discuss the recent development on planar graph cut algorithms [2, 22] and propose an accelerated version by reducing the amount of involved data structures. In Section 4, we will present experimental benchmarks on image segmentation and shape matching. We will show that this method provides the first graph cut based shape matching method which needs only $O(N^2 \log N)$ computation steps. Additionally, we provide an $O(N \log N)$-approach for image segmentation. In both experiments our algorithm exhibits runtimes which are not only consistently smaller than those of the Boykov and Kolmogorov algorithm, but which are also more predictable in the sense that they exhibit a far smaller spread. Section 5 provides a conclusion.

2. Polynomial Time Solutions for Graph Cuts

In this section, we present some basic notations for graph cuts. This general formalism will be used to solve the shape matching and image segmentation task in Section 4. A graph network $G = (V, E, c, s, t)$ consists of a set of vertices V that are connected via oriented edges $E \subset V \times V$ whereas the source $s \in V$ provides only outgoing edges and the sink $t \in V$ only incoming edges. Every edge $e = (u, v)$ is equipped with a positive capacity $c(e) \in \mathbb{R}^+$ and we call $C \subset E$ an st-cut if there is no path from the source $s \in V$ to the sink $t \in V$ in the reduced network $\widetilde{G} = (V, (E \setminus C), c, s, t)$. The problem of finding a *minimal cut* C can be formulated as following:

Problem 1 MINIMAL CUT
Input: Graph network $G = (V, E, c, s, t)$
Output: st-Cut $C \subset E$ which minimizes $\sum_{e \in C} c(e)$

A major milestone to solve this general problem was the MinCut-MaxFlow theorem [9] which stated that the MINIMAL CUT problem is equivalent to solving the MAXIMAL FLOW problem:

Problem 2 MAXIMAL FLOW
Input: Graph network $G = (V, E, c, s, t)$
Output: Flow $f \leq c$ maximizing $\sum_{(v,t) \in E} f(v, t)$

The main idea to solve the MAXIMAL FLOW problem is to start with a flow $f : E \to \mathbb{R}_0^+$ that assigns 0 to every edge $e \in E$ and then to augment the flow along paths from source to sink on which none of the involved edges are saturated, *i.e.*, $f(e) < c(e)$. This *augmenting path strategy* solves the problem and if we are only using shortest paths, the problem can be solved in polynomial time [7]. Nonetheless, this runtime is in general too high. In [4] the authors presented a min-cut algorithm solving many Computer Vision problems effectively in linear time. However, the approach cannot guarantee to always augment the shortest path from source to sink and therefore it provides no polynomial upper runtime bound.

On the other hand, it is known [22] that an almost linear runtime[1] is an upper bound for planar networks. These are networks that can be embedded in the plane $\mathbb{R}^2$. This means that the edges cut the plane open into different faces F. An import property of planar graphs is that two different edges may *not cross* each other. Therefore, every edge e has a well defined left face $f_l(e) \in F$ and right face $f_r(e) \in F$. To every planar network G, we can therefore define the dual network $G^* := (F, E^*, c, s^*, t^*)$ by introducing dual edges $e^* := (f_l(e), f_r(e))$ which connect the left with the right face of an edge. The dual vertices s^* and t^* are arbitrary faces in the vicinity of s and t respectively. In the next section, we will show that the planar graph cut needs almost linear runtime by revising and improving the work of [2]. In Section 4 we will apply this method to Computer Vision problems and present a substantially runtime acceleration for shape matching and image segmentation.

3. Efficient Planar Graph Cuts

In [22], it was first shown that for planar graphs with N vertices, the maximal flow can be computed in $O(N \log N)$. Unfortunately, the proposed method requires a rather complicated preprocessing step and hence is not ideally suited for practical implementations. To overcome this drawback, in [2] a new method was proposed that uses a simpler preprocessing step. The core idea of this method is it to augment always the *leftmost* of all paths from source to sink:

Algorithm 1 Planar Maximal Flow [2]
Input: Planar Graph network $G = (V, E, F, f_l, f_r, c, s, t)$
Output: Maximal Flow $f : E \to \mathbb{R}^+$
1: Remove from G all clockwise cycles
2: Initialize the flow f with 0
3: **while** there is a non-saturated path from s to t **do**
4: saturate the leftmost path from s to t
5: **end while**
6: return f

Step 1 of Algorithm 1 is a preprocessing step which can be computed in $O(N \log N)$. A challenging implementation task is in fact Step 4. Like any augmenting path method, it (cf. Algorithm 2) handles a spanning tree T of all vertices V to keep track of the augmenting path from source to sink efficiently. Additionally, the method handles also a spanning tree T^* of all faces F to support the updating scheme of T. Since the graph is planar, all edges which are not in T form the tree T^* (cf. Figure 2). Lines 8, 9 and 13 of Algorithm 2 take care of this invariant.

[1] By *almost*, we are referring to the $\widetilde{O}$-notation, *i.e.*, $n \log(n) = \widetilde{O}(n)$.

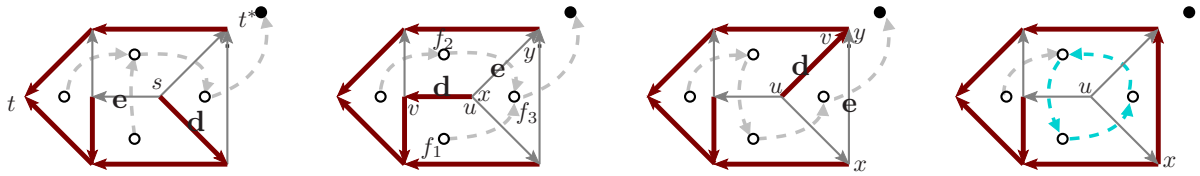

Figure 2. Planar max flow method.(A|B|C|D):In each step, the thick edges indicate the spanning tree T of the primal graph. Dashed lines represent the edges of the spanning tree T^* of the dual graph. **A** shows the initialization of the method. At every step, an edge $d = (u, v)$ of the tree T is substituted by an edge $e = (x, y)$. When the method terminates in **D**, neither T nor T^* are trees anymore. Moreover, in T^* a circle emerges which describes the minimal cut.

Algorithm 2 Implementation of Step 4 [2]

1: Let T be the right-first search tree backward from t.
2: Let T^* be the spanning tree of F consisting of all edges of $E - T$.
3: **repeat**
4: Augment path from s to t, update the flow f and let $d = (u, v)$ the closest edge to t which is saturated.
5: Let (f_1, f_2) the dual edge d^* of d.
6: Let $e^* = (f_2, f_3)$ be the parent edge of f_2 in T^*.
7: Let $e = (x, y)$ be the primal edge with respect to e^*.
8: $T^* := T^* + \{(f_2, f_1)\} - \{(f_2, f_3)\}$.
9: $T := T - \{d\} + \{e\}$.
10: **if** f_1 is a descendent of f_2 within T^* **then**
11: return f
12: **end if**
13: Reverse in T the edges along the path from x to u.
14: **until false**

It is shown that the **repeat**-loop is repeated at most $O(N)$ times and that the usage of Dynamic Tree [20] for T and Euler Tour Tree [11] for T^* results in an $O(\log N)$-runtime for the Lines 4-13. Hence, the method computes the maximal flow in $O(N \log N)$.

Nonetheless, the test for the termination condition (Line 10) is a bottleneck of Algorithm 2. The theoretical contribution of our work is to get rid of the Euler Tour Tree and instead maintain T^* by an array which stores the parent of each face. Originally, the Euler Tour Tree was used in order to test Line 10 in $O(\log N)$. However, modification and parent access (Line 6 and 8) of T^* then take the same amount of time. We therefore propose an equivalent test on T instead of T^*. In this section, we will present this alternative test and prove its equivalence in Theorem 1. Instead of Lines 10-12, we perform the test of Algorithm 3:

Algorithm 3 Alternative Termination Condition for T

10: **if** there is no path from x to u in T **then**
11: return f
12: **end if**

Surprisingly, the new test takes no additional time, since the path from x to u has to be identified in Line 13 anyway. Furthermore, the T^*-related Lines 6 and 8 can now be done in $O(1)$ instead of $O(\log N)$. This runtime reduction makes this method attractive for Computer Vision tasks as we will see in Section 4. The following theorem proves the correctness and efficiency of the proposed method:

Theorem 1. *The proposed method solves the Maximum Flow problem in $O(N \log N)$.*

Proof. Our approach substitutes Lines 10–12 of Algorithm 2 with Algorithm 3. As long as f_1 is not a descendant of f_2, both approaches do not differ from one another. This is due to the fact that Line 13 of Algorithm 2 implies the existence of a path from x to u in T. Therefore, let us now assume that f_1 is in fact a descendant of f_2. Then, there exists a path from f_1 to f_2 in the dual tree T^*. In Line 8 the dual edge (f_2, f_1) is inserted into T^* and this data structure possesses now a counter clockwise circle which encloses the vertex u (cf. Figure 2,D). Since e^* was an edge that left f_2, the two vertices x and y are now outside of the just constructed circle. Therefore, Line 10 of Algorithm 3 cannot find a path from x to u and the proposed method terminates returning the same flow as Algorithm 2. The computational complexity is preserved. □

4. Applications

In Computer Vision there is an abundance of problems that can be addressed as finding a minimal cut through a planar graph such as *shape matching*, *image segmentation* or *cyclic time series*. In this section, we address the first two problems which are chosen exemplarily to substantiate the relevance of planar graphs in Computer Vision.

4.1. Shape Matching

An important task of Computer Vision is to classify a given contour $c_0 : \mathbb{S}^1 \to \mathbb{R}^2$ which maps every point of the parameterizing circle $\mathbb{S}^1$ onto the plane $\mathbb{R}^2$. To perform this classification task, we measure the similarity between the given contour and a collection of known shapes. A shape $\mathcal{C}$ consists of a whole class of different curves $c : \mathbb{S}^1 \to \mathbb{R}^2$ which is invariant under rigid body transformations $\mathrm{SE}(2)$.

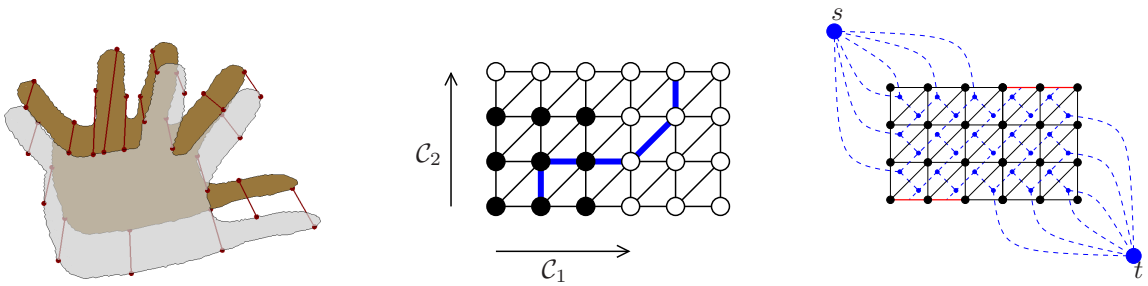

Figure 3. Planar Shape Matching. *Left:* To measure the dissimilarity of two given shapes, we are looking for corresponding points on different shapes. *Middle:* By sampling two shapes $\mathcal{C}_1$ and $\mathcal{C}_2$ with N points, we receive a squared graph (filled vertices). Every matching can be represented by a path from $(a, 0)$ to $(a + N, N)$. *Right:* Identifying these vertices, results in glewing the red edges together. Every cycle on this arising cylinder describes a cut on the dual graph G^* (dashed edges).

To measure the dissimilarity of two given shapes, we are looking for a dynamic correspondence mapping $m : \mathbb{S}^1 \to \mathbb{S}^1$ that maps any point of the first shape onto the corresponding point of the second shape. Hence, we are interested in minimizing the energy functional

$$E_{f_1,f_2}(m) := \int_{\mathbb{S}^1} \|f_1(s) - f_2(m(s))\|^2 \,\mathrm{dm}(s), \qquad (1)$$

where f describes a feature that is $\mathrm{SE}(2)$-invariant. Besides the curvature, the *shape context* [1] and the *inner shape context* [14] are prominent features for shape analysis.

4.1.1 Shape Matching via Dynamic Programming

It is well known that the energy functional (1) can be minimized by finding the shortest path in a specific graph (cf. middle of Figure 3). Any vertex $(x, y) \in V$ represents a possible match between $c_1(x)$ on the first shape and $c_2(y)$ on the second shape and the data term of this vertex is $\|f_1(x) - f_2(y)\|^2$. Therefore, the weight of an edge $(x_1, y_1) \to (x_2, y_2)$ carries the line integral along this path. If we sample each shape by N points, any path from $(x, 0)$ to $(x + N, N)$ describes a valid matching. Hence, the minimizer of (1) can be found by looking for the shortest path in that graph. Given an initial correspondence $(x, 0)$, the shortest path can be computed via Dynamic Programming [14] which takes linear time in the size of the graph. After testing thoroughly ever possible initial correspondence, we would end up with a runtime of $O(N^3)$. It is possible to reduce the runtime for this specific application [17]. But the proposed graph cut approach reduces the time complexity directly as we will show below.

4.1.2 Shape Matching via Graph Cuts

It is possible to transform the shape matching problem into a graph cut problem: By identifying every node $(a, 0)$ with $(a+N, N)$, the graph becomes a cylinder and each shortest path becomes a closed cycle that separates the two outmost nodes (cf. Figure 3). After taking the dual graph [18], every cut corresponds to a valid matching between the compared shapes. The matching corresponding to the minimal cut represents the minimizer of functional (1). Since the constructed graph is planar, we can apply the presented graph cut method of Section 3. The size of the graph is $O(N^2)$ and the proposed method runs in $O(N^2 \log N)$. In Figure 5, the runtime for two different examples are given. As we can see, the presented implementation outruns the other methods and provides a speed-up factor of 2–4 with respect to the the original work of Borradaile. Interestingly, the graph cut approaches become faster at the presence of similar shapes. This is due to the fact that the cut, *i.e.*, the bottleneck of the defined network, is much easier to detect if we compare similar shapes.

The efficiency of shape matching is crucial to analyze shapes correctly. A very challenging database is the MPEG7-database (cf. Figure 6). It consists of 70 different shape classes which are represented by 20 different shapes each. For the benchmark, one calculates the 40 closest shapes to a given shape according to the used metric. If among these shapes, there are k shapes of the same class as the query shape, the retrieval rate is $\frac{k}{20}$. The mean of all 1400 retrieval rates is the retrieval rate of the database. There exist several methods that were used to handle this database. Among the best is the *graph transduction approach* [23] with a retrieval rate of 91% which is based on the *inner shape context* [14]. While maintaining the same retrieval rate, our algorithms allows to reduce the computation time for the inner shape context. Figure 5 indicates that the speed-up factor of the proposed method increases substantially for increasing shape resolutions.

4.2. Image Segmentation

A second computational challenge in Computer Vision is that of image segmentation. A classical approach is the *geodesic active contour* pioneered in [5]. This approach

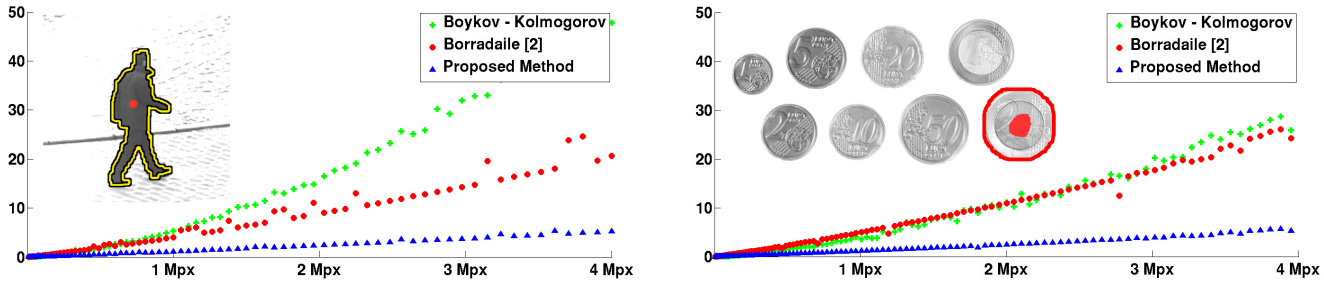

Figure 4. Runtimes of image segmentation. The runtime on the y-axis is measured in seconds and depends on the resolution of the images. The proposed approach is faster than the method of Boykov and Kolmogorov. Also, it has a much smaller runtime spread for different images and the observed speed-up increases at higher image resolutions.

tries to minimize energies of the form

$$E(C) := \int_C g(|\nabla I|)\mathrm{dC} + \alpha\,\mathrm{length}(C), \qquad (2)$$

with respect to a segmentation boundary C and a monotonically decreasing function g that penalizes low image gradient along that boundary. Therefore, the minimizing curve is forced to areas of the image which contain a high image gradient. But unfortunately, the global minimum of (2) is the empty contour. This drawback can be overcome by restricting the set of feasible curves which leads to interactive segmentation tools:

- The user may select a certain region of the image that should belong to the object and the curves that shall minimize (2) are then restricted to those that wind the given region exactly once [8].
- The user may suggest an object's boundary and the set of feasible curves is restricted to those which lie within the vicinity of the input curve. This approach is also known as *intelligent scissor* [15].

Both interactive approaches select a certain connected region S as object and another region T as background. By replacing S and T with nodes s and t, respectively, the minimizer of (2) is a closed circular path which winds around s exactly once. For the given planar graphs, this is equivalent to an st-cut. Therefore, this specific segmentation task can be solved by a minimal cut approach. If the input image consists of N pixel, the segmentation task can be performed in $O(N \log N)$ calculation steps by using the proposed method.

In Figure 4, the result of a benchmark test is plotted and one can see that the presented maximum flow approach outruns the method of Boykov and Kolmogorov. For the benchmark we used different images that consist of approximately 4 Megapixels each. These images were scaled down to different resolution and we tested both methods on the smaller versions. We can therefore compare how the two methods work on the same scene with different resolutions. Figure 4 indicates that the runtime of the proposed method is more predictable than the method of Boykov and Kolmogorov. Since the speed-up factor increases with an increasing image resolution, the proposed method is also more suited for high-resolution images. As in the shape matching example, the speed-up factor of the proposed approach with respect to the original work is 2–4.

5. Conclusion

In this paper, we presented a graph cut approach for planar graphs which solves the problem in $O(N \log N)$ and thus – in contrast to the currently used algorithm of Boykov and Kolmogorov – provides a known upper bound on the worst case complexity. Our algorithm is built on the recently published method [2]. It reduces substantially the amount of required data structures and is therefore faster. We demonstrated the advantages of this compressed version of a maximum flow method in two benchmark tests, one on planar shape matching with increasing shape resolution, and one on image segmentation with increasing image resolution. For these examples, we observed a speed-up factor of 2–4 w.r.t. the work of Borradaile and of an order of magnitude w.r.t. the work of Boykov and Kolmogorov. Hence, the proposed graph cut method outperforms the one of Boykov and Kolmogorov in terms of worst-case complexity, in terms of actual (experimental) runtimes and in terms of predictability (observed spread of runtimes). This makes our algorithm well suited for time-critical applications where a predictable and guaranteed fast performance is required.

References

[1] S. Belongie, J. Malik, and J. Puzicha. Shape matching and objects recognition using shape contexts. *IEEE PAMI*, 24(4):509–522, 2002.

[2] G. Borradaile. *Exploiting Planarity for Network Flow and Connectivity Problems*. PhD thesis, Brown University, May 2008.

[3] Y. Boykov and M.-P. Jolly. Interactive organ segmentation using graph cuts. In *MICCAI*, volume 1935 of *LNCS*, pages 276–286. Springer, 2000.

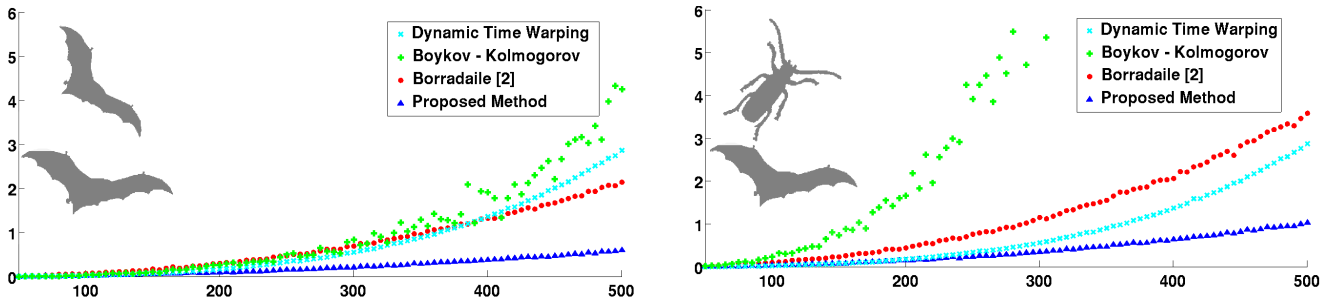

Figure 5. Runtimes of Shape Matching. The runtime on the y-axis is measured in seconds and depends on the amount of shape points of each shape. The proposed method outruns the existing method of Boykov and Kolmogorov and is also faster than Dynamic Programming. The speed-up factor increases with an increasing shape resolution.

Figure 6. MPEG7-Retrieval. While for shape classes that are easily confused, retrieval rates can drop below 50% (middle row), the average retrieval rate is at 85%. The proposed graph cut method allows to substantially accelerate the required shape distance computation.

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

[5] V. Caselles, R. Kimmel, and G. Sapiro. Geodesic active contours. In *IEEE ICCV*, pages 694–699, Boston, USA, 1995.

[6] A. Delong and Y. Boykov. A scalable graph-cut algorithm for n-d grids. In *IEEE CVPR*, Anchorage, Alaska, June 2008.

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

[8] D. Farin, M. Pfeffer, P. H. N. de With, and W. Effelsberg. Corridor Scissors: A semi-automatic segmentation tool employing minimum-cost circular paths. In *IEEE ICIP*, Singapore, October 2004.

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

[10] D. M. Greig, B. T. Porteous, and A. H. Seheult. Exact maximum *a posteriori* estimation for binary images. *J. Roy. Statist. Soc., Ser. B.*, 51(2):271–279, 1989.

[11] M. R. Henzinger and V. King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46(4):502–516, 1999.

[12] O. Juan and Y. Boykov. Capacity scaling for graph cuts in vision. In *IEEE ICCV*, 2007.

[13] P. Kohli and P. H. S. Torr. Dynamic graph cuts for efficient inference in markov random fields. *IEEE PAMI*, 29(12):2079–2088, 2007.

[14] H. Ling and D. W. Jacobs. Shape classification using the inner-distance. *IEEE PAMI*, 29(02):286–299, 2007.

[15] E. N. Mortensen and W. A. Barrett. Intelligent scissors for image composition, 1995.

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

[17] F. R. Schmidt, D. Farin, and D. Cremers. Fast matching of planar shapes in sub-cubic runtime. In *IEEE ICCV*, Rio de Janeiro, Brazil, October 2007.

[18] F. R. Schmidt, E. Töppe, D. Cremers, and Y. Boykov. Efficient shape matching via graph cuts. In *Int. Conf. on EMMCVPR*, 2007.

[19] S. N. Sinha and M. Pollefeys. Multi-view reconstruction using photo-consistency and exact silhouette constraints: A max-flow formulation. In *IEEE ICCV*, pages 349–356, 2005.

[20] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. In *JCSS*, pages 362–391, 1981.

[21] G. Vogiatzis, P. Torr, and R. Cippola. Multi-view stereo via volumetric graph-cuts. In *IEEE CVPR*, pages 391–399, 2005.

[22] K. Weihe. Maximum (s,t)-flows in planar networks in O($|V|$ log$|V|$) time. *J. Comput. Syst. Sci.*, 55(3):454–475, 1997.

[23] X. Yang, X. Bai, L. J. Latecki, and Z. Tu. Improving shape retrieval by learning graph transduction. In *Europ. Conf. on Computer Vision*, pages 788–801, Marseille, France, October 2008.