Proceedings of the Czech–Japanese Seminar in Applied Mathematics 2010
Czech Technical University in Prague, August 30 - September 4, 2010
pp. 46–54

# GRAPH CUTS IN SEGMENTATION OF A LEFT VENTRICLE FROM MRI DATA

JAKUB LOUCKÝ[1] AND TOMÁŠ OBERHUBER[1]

**Abstract.** The article presents applications of graph cuts in the image processing. We describe the original Ford–Fulkerson algorithm and its modification — Edmonds–Karp algorithm. We also present our modifications of this algorithm. These modifications lead to a major acceleration of the segmentation process, which is demonstrated using real images of a left ventricle. Although the results are not equivalent to the non-modified algorithm, this modification is suitable for our application.

**Key words.** image segmentation, graph cuts, Ford–Fulkerson algorithm

**AMS subject classifications.** 05C21, 68R10, 68U10, 90C27

**1. Introduction.** This paper refers to significant former results in the field of image segmentation using graph cuts. We show the relation between graph cuts and resulting image segmentation. We present a classic algorithm for finding the *maximal flow* in a directed weighted graph, which leads to the *minimal cut* and can be applied in the semi-automatic segmentation of an image. We also present our own modifications of this algorithm. Finally we compare implementations of both the original Edmonds–Karp (slightly modified Ford–Fulkerson) algorithm and the modified algorithm. We test them by segmenting images of a left ventricle from MRI data.

Graph cuts in image processing were first introduced in [8] for binary image reconstruction. Since then, similar methods were developed and applied for various tasks including image segmentation. Details on the initial graph construction can be found in [2], [4] and [9]. Authors also describe methods of reusing a former result during new computation with new initial conditions added.

Fast algorithms for finding maximal flow in a weighted graph are required in all these applications. The first approach to a maximum flow problem using the graph theory was presented in [5]. Many modifications followed, forming a group of *augmenting path* methods. Later, [7] introduced different approach, which created a second group of *push-relabel* methods. Further experimental results are presented in [1]. In [3] authors introduce their algorithm combining approach from both augmenting path and push-relabel groups. They compare their algorithm with various existing algorithms.

Further improvements of image segmentation methods are described in [6] and [10]; the former uses prior information about the object shape, the latter presents a method enforcing connectivity of the segmented object.

**1.1. Organization.** First, in the Section 2 we introduce a *s–t cut*. The Section 3 explains the process of creating a graph based on a given image. Here we show that an appropriate approach to the graph construction leads to a correspondence

[1]Department of Mathematics, Faculty of Nuclear Sciences and Physical Engineering, Czech Technical University, Prague.

between minimal graph cut and object segmentation. In the Section 4 we present the duality of minimum cut and maximum flow problems. Furthermore, we describe the classic Ford–Fulkerson algorithm for finding the maximum flow. The Section 5 introduces our modifications of the original algorithm. These modifications lead to a major speed-up of the segmentation process. As a consequence the segmentation results are altered, but they are even more suitable for our application. Finally, we present the results of our modified algorithm and compare them with the original algorithm in the Section 6.

**1.2. Contributions.** We implement Edmonds–Karp algorithm for the image segmentation and perform modifications of the algorithm. These modifications combined together lead to major speed-up of the segmentation process. Tests using real images of a human heart show 50× to 100× improvement compared to original algorithm. Further improvements could lead to an interactive segmentation in real time.

**2. Graph cuts in image segmentation.** Let $G = (V, E, w)$ be a directed weighted graph (network). $V$ denotes a set of its nodes, $E$ is a set of directed edges connecting two nodes, $w : E \to \mathbb{R}_0^+$ is a cost function that assigns each edge a non-negative value – *capacity*. The structure of a example graph is illustrated in figure 2.1.

The set $V$ includes two special nodes: *source* $s$ and *sink* $t$. These two special nodes are referred to as *terminal nodes*, $P$ denotes the remaining non-terminal nodes; $V = P \cup \{s, t\}$.

Edges from the set $E$ connect either a terminal node with a non-terminal node or two non-terminal nodes. Nodes $s$ and $t$ are not connected by any edge.

A *s–t cut* $C = \{S, T\}$ is a partitioning of $V$ into two disjoint subsets $S$ and $T$ so that $s \in S$ and $t \in T$. The cost of the cut $C$ is the sum of costs of all edges that connect a node from $S$ with another one from $T$. Edges directed from $S$ to $T$ are added with positive sign, edges directed in the opposite direction have negative sign.

Our goal will be to find the *minimum s–t cut*, the cut with the minimal cost. As shown in the Section 3, by choosing an appropriate cost function $w$ the minimum s–t cut will be equivalent to the image segmentation. Subsets $S$ and $T$ will correspond to the object segment and background segment, respectively.

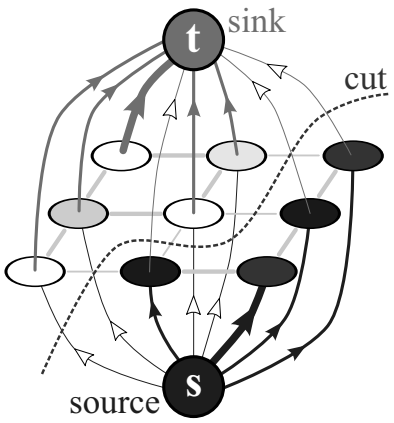



FIG. 2.1. *The structure of a example graph.*

**3. Construction of the graph.** Let's presume we are given a grayscale image as a rectangular 2D grid of pixels. It can be represented as a $m$-by-$n$ matrix $I$, for each $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$ the matrix element $I_{ij}$ is an integer and denotes the intensity of a pixel with coordinates $[i, j]$. The image $I$ needs to be segmented into two parts: an object and its background. Some image pixels are marked as *object seeds*, i.e. pixels that are supposed to be a part of the object segment. Similarly some other image pixels are marked as *background seeds*.

We start by the construction of a weighted directed graph with two *terminal nodes*: $s$ and $t$. Any s–t cut represents partitioning of the given image into two parts; pixels corresponding to nodes in subset $S$ form one part of the image, remaining pixels (corresponding to nodes in subset $T$) form the other part. By evaluating edge capacities appropriately the minimum s–t cut will represent segmentation of the given image into object segment and background segment.

The set of *non-terminal* nodes $P$ consists of the same number of nodes as is the count of image pixels. Each node corresponds to one image pixel. Nodes corresponding to object and background seeds form sets $O$ and $B$, respectively. $O \subset P$, $B \subset P$. One pixel cannot be a part of both object and background, thus $O \cap B = \emptyset$.

The set of edges consists of *t-links* and *n-links*. The *t-links* are directed edges $(s, p)$ and $(p, t)$ for each $p \in P$. Every *t-link* is assigned a capacity which expresses the similarity of each individual pixel to object seeds or background seeds (referred to as the *regional property*). Similarity is measured as the difference of intensities. The *n-links* are directed edges $(p, q)$, where $p, q$ correspond to two neighboring pixels. Various (e.g. 4- or 8-) neighborhood systems can be used. Here we use 4-neighborhood system, so each pixel $I_{ij}$ has up to 4 neighbors $I_{i+1,j}$, $I_{i,j+1}$, $I_{i-1,j}$ and $I_{i,j-1}$. Pixels on the border (for $i = 1 \vee j = 1 \vee i = m \vee j = n$) have only 3 or 2 neighbors. There are no *n-links* connecting two non-neighboring pixels. These *n-links* enforce a certain level of smoothness of the segmentation result (referred to as the *boundary property*).

The next step is to evaluate the edge capacities of this graph with respect to our demand that the minimum s–t cut divides nodes into an object segment and a background segment. Therefore edges within the object and edges within its background should have capacities large enough not to be severed by the minimum cut. On the contrary edges near the border between the object and its background need to have their capacities small. General required properties of the cost function $w$ will follow.

First *t-links* with the seeds need to reflect the fact that the set of object seeds $O$ is included in the object and not in the background, similarly for the set $B$. For each $p \in O$ the cost of $(s, p)$ is set to infinity (or the highest possible number) while the cost of $(p, t)$ is zero. For each $p \in B$ the cost of $(s, p)$ is set to zero while the cost of $(p, t)$ is infinite.

Other *t-links*, which are not connected to seeds, will describe the similarity of given pixels to seeds. Edges $(s, p)$ express the similarity to object seeds while edges $(p, t)$ express the similarity to background seeds. The closer the intensity of given pixel and the intensity of seeds are, the larger the capacity of the appropriate edge will be.

The capacity of *n-links* is related to the similarity of neighboring pixels. The closer their intensities are, the larger the capacity needs to be. Near the border between the object and its background largest intensity differences between neighboring pixels (and therefore edges with smallest capacities) are expected.

As the costs of *t-links* and *n-links* are not necessarily related, we adjust a relative importance of the regional property versus the boundary property by the introduc-

| Type | Edge | | Cost |
|---|---|---|---|
| *n-link* | $(p,q)$ | for $p,q \in P$ | $N(p,q)$ |
| *t-link* | $(s,p)$ | for $p \in P \setminus \{O \cup B\}$ | $\lambda R_s(p)$ |
| | | for $p \in O$ | $\infty$ |
| | | for $p \in B$ | 0 |
| | $(p,t)$ | for $p \in P \setminus \{O \cup B\}$ | $\lambda R_t(p)$ |
| | | for $p \in O$ | 0 |
| | | for $p \in B$ | $\infty$ |

TABLE 3.1
*Costs of edges.*

tion of $\lambda$-weighted combination of these two properties. Let the costs of *t-links* are multiplied by $\lambda$. By choosing a large $\lambda$ parameter we prefer regional property while small $\lambda$ enforces a smoother border.

Table 3.1 sums up the procedure of assigning costs to edges as presented above. $N(p,q)$ denotes the cost function for an *n-link* $(p,q)$, $R_s(p)$ and $R_t(p)$ are cost functions for *t-links* $(s,p)$ and $(p,t)$, respectively.

We have chosen simple linear cost functions depending on a difference of pixel intensities as follows:

$$N(p,q) = D - |I_p - I_q|,$$

where $I_p$, $I_q$ denote the intensity of the pixels $p$, $q$ and $D = I_{\max} - I_{\min}$ is a constant that keeps function $N$ non-negative;

$$R_s(p) = M - \left|\overline{I_s} - I_p\right|,$$
$$R_t(p) = M - \left|\overline{I_t} - I_p\right|,$$

where $\overline{I_s}$, $\overline{I_t}$ denote the average value of intensities of object seeds and background seeds, respectively; $M$ is the maximum possible value of a pixel intensity. Functions $R_s$ and $R_t$ are also non-negative. These functions satisfy general requirements presented in the previous paragraphs.

If $(p,q)$ is an *n-link*, $(q,p)$ is also an *n-link* and $N(p,q) = N(q,p)$. Therefore each pair of edges $(p,q)$ and $(q,p)$ can be replaced by only one of these edges. While the values of allowed flow in both former edges were from $[0, N(p,q)]$, the new edge will allow also negative flow: $[-N(p,q), N(p,q)]$. This change perceives the former symmetry and simplifies the implementation.

**4. Finding the minimum s–t cut.** The theorem of Ford and Fulkerson [5] states that a maximum flow from $s$ to $t$ saturates a set of edges corresponding to a minimum s–t cut. Moreover, the cost of the minimum cut is equal to the value of the maximum flow. Thus, the minimum cut and the maximum flow are dual problems. By finding the maximum flow we receive the minimum cut.

The Ford–Fulkerson algorithm is a basic algorithm that solves the maximum flow problem. It belongs to the group of *augmenting path* algorithms. The augmenting path is a path from $s$ to $t$ that has available capacity on all its edges. The main idea is that we start with zero flow and as long as an augmenting path exists, we push

the maximum possible flow along it – we *saturate* it. By saturating each augmenting path the total flow in the graph is increased. Once no augmenting path exists, the maximum flow is reached and the algorithm terminates.

**4.1. Marking procedure.** This procedure performs a search for an augmenting path. Let $f(e)$ denote current flow in the edge $e$, $c(e)$ is the capacity of $e$.

1. Mark source node $s$, all other nodes are not marked.
2. $(\exists e = (p, q) \in E$, $p$ is marked, $q$ is not marked, $f(e) < c(e)) \Rightarrow$ mark $q$
3. $(\exists e = (p, q) \in E$, $p$ is not marked, $q$ is marked, $-c(e) < f(e)) \Rightarrow$ mark $p$
4. If $t$ is marked, an augmenting path exists. STOP.
   If no node has been marked since last loop iteration, no augmenting path exists. STOP.
5. GOTO 2.

Steps 2 and 3 can be implemented in various ways. Edmonds–Karp algorithm uses breadth-first search, which results in finding the shortest augmenting path in each step.

For the reconstruction of the augmenting path we need to record which node caused the marking and in which direction; step 2 "marks forwards", step 3 "marks backwards". Then, the augmenting path can be reconstructed by tracing back from $t$ to $s$.

**4.2. Tracing back and saturating the augmenting path.** When the sink $t$ has been marked, we reconstruct the augmenting path. Simultaneously we find the maximum value of flow which can be added to the former flow along this path. This increment is strictly positive. It is added to the flow in edges "marking forwards" and is subtracted from the flow in edges "marking backwards".

**4.3. Ford–Fulkerson algorithm.**

1. Perform the marking procedure (4.1).
   If no augmenting path doesn't exist, GOTO 3.
2. Trace back and saturate the augmenting path (4.2)
   GOTO 1.
3. Maximum flow has been reached.
   Let $S$ be a set of all nodes that have been marked by the last marking procedure in step 1. Let $T = V \setminus S$. $\{S, T\}$ is the minimum s–t cut.

Generally the Ford–Fulkerson algorithm is not always finite. But assuming finite integral (or rational) capacities of all edges, the algorithm finishes in finite number of steps. Each step increases the total flow by a positive value, in case of integral capacities by at least 1. The maximum flow is finite, thus only finite number of augmenting paths will be found.

**5. Modifications of the algorithm.** Here we present two major changes in the implementation of Edmonds–Karp algorithm. The first one is easy to perform, significantly reduces running time with graphs based on real data and does not change the result compared to non-modified algorithm.

Second modification further improves the speed of the algorithm. While the resulting segmentation is no longer the same, we show that it corresponds better to the expectations in our application.

**5.1. Improvement of the marking procedure.** The original marking procedure (4.1) stops right after the sink has been marked. Then we are able to reconstruct only one augmenting path. However if we allow multiple marks of the sink, more than one augmenting path can be found at once after performing one marking procedure. As a breadth-first search is used, all these paths found in one step have the same length.

Because the marking procedure is a very time consuming operation, this modification reduces the total count of marking procedures before the algorithm terminates.

Note that now not all of these paths found in one step will actually increase the total flow in the graph. The reason for this is that multiple paths can share some of their edges. There is a chance that by saturating one path we also saturate others.

In the worst case only one of these paths will increase the total flow while others will not; this modification would not bring any advantage in this case. But tests with graphs constructed in our application show that the implementation modified in this way completes the segmentation process about ten times faster than the original algorithm while the results of both implementations are identical.

**5.2. Further modifications.** The original algorithm used for image segmentation starts by marking the source. In the next step it marks all nodes connected to the source by a non-saturated edge, which often means vast majority of the set $P$. So the algorithm spends a lot of time in those parts of the image that are far from expected result. Following alternation of the algorithm significantly reduces this problem.

First, we define a new function that will help to describe the alternated algorithm. It is clear that for all $p \in P$ a path only consisting of two *t-links* $(s,p),(p,t)$ is the augmenting path if neither of these links is saturated. The saturation of all these paths will always be the first step of the algorithm and leads to the saturation of at least one *t-link* in each path. At most one *t-link* to each node $p$ preserves free capacity.

For all $p \in P$ we define a new function $R : P \to \mathbb{R}$ as follows:

$$R(p) := \lambda R_s(p) - \lambda R_t(p).$$

This function fully describes the state of all *t-links* after previously introduced first step of the algorithm. A positive value means that the *t-link* from the source has some remaining capacity $R(p)$ while the *t-link* to the sink is fully saturated. A negative value indicates that the *t-link* from the source is saturated while the *t-link* to the sink preserves capacity $|R(p)|$. $R(p) = 0$ means that both *t-links* got saturated. We evaluate this function during the construction of the graph.

Now we propose to start the marking procedure from all object seeds instead of the source $s$. The procedure will end by marking any $p \in P$ so that $R(p) < 0$. Because the starting node is connected to the source by a non-saturable edge (with infinite capacity) and the node at the end is connected to the sink by a non-saturated edge, we have found one augmenting path from the source to the sink. However, this path never includes *t-links* from the source with a finite capacity. These links would be meaningless. Therefore we also modify the procedure reconstructing the augmenting paths.

Instead of one path we reconstruct more of them. Each of them starts in the source, follows a different *t-link*, then they unite one after another and finally they reach the sink together using common *t-link*. They form a system of paths that can increase the total flow at most by the value $-R(p)$, where $p$ was the last node marked at the end of the marking procedure. While following the path back we bring flow

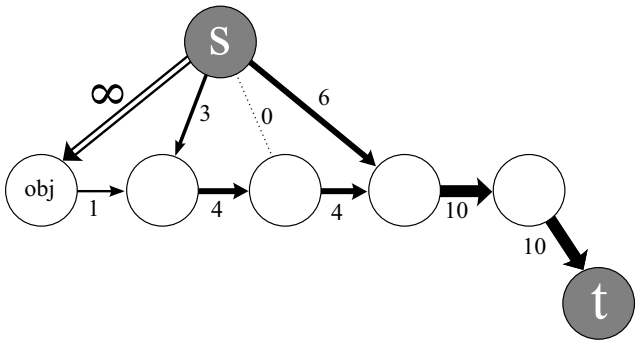



FIG. 5.1. *Reconstruction of the system of paths.*

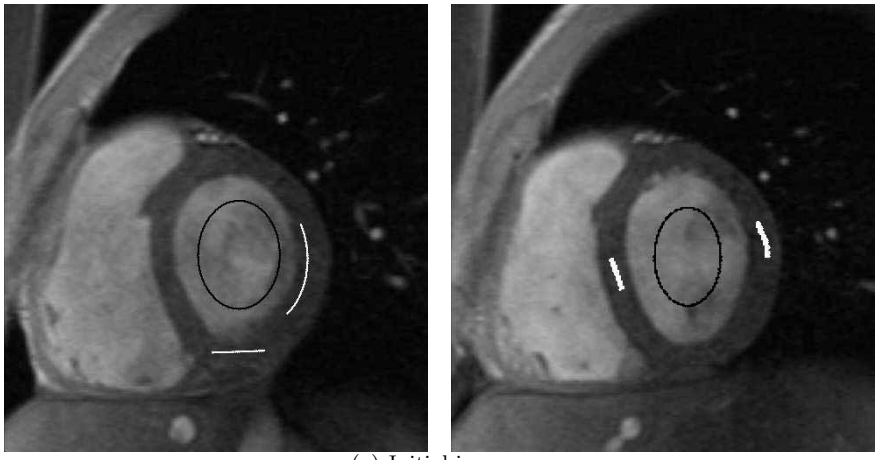
using *t-links* until we reach the object seed or the total flow of this system reaches the maximum value $-R(p)$.

Figure 5.1 shows an example of this reconstruction; "obj" denotes an object seed. We start with the last non-terminal node on the right. This system of paths can increase the total flow at most by 10. Previous node is connected to the source by an edge with capacity 6. We have found the first augmenting path and only the flow of size 4 remains. The foregoing node has its *t-link* fully saturated, so we carry on. The second node in line has a *t-link* with free capacity 3, we have found second augmenting path. The flow of size 1 is brought along the third, longest augmenting path through the object seed. We have increased the total flow by 10 altogether and the *t-link* to the sink got saturated.

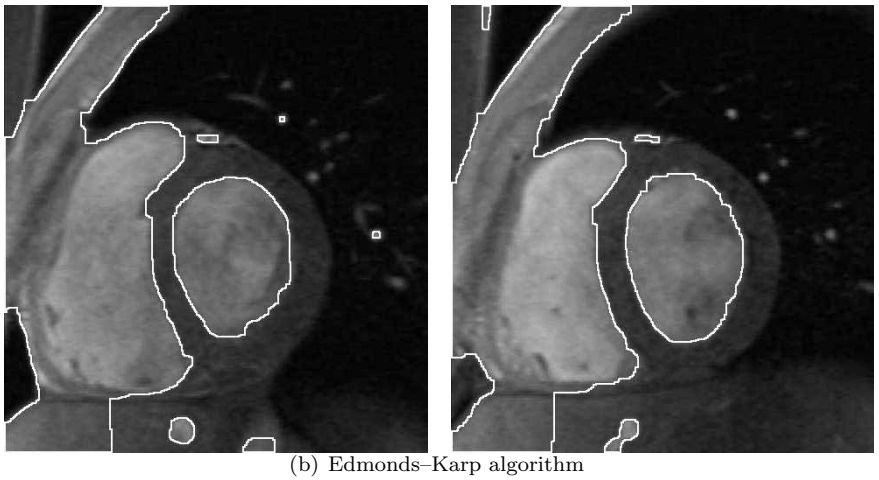
Results of the algorithm modified in this way will differ from the unchanged algorithm. In general we perform the marking in a smaller area, thus resultant object segment will be smaller, especially in the case of the result consisting of several disjoint areas. The resultant segmentation of the original algorithm can contain more disjoint segments even if there was only one object seed. Our modified algorithm requires at least one object seed in every disjoint segment, otherwise marking would neither start in this segment nor reach it. This requirement is not a limitation in our application, our goal is to segment only one object.

**6. Results.** We have implemented two variants of Edmonds–Karp algorithm for image segmentation: the first implementation incorporates only the simple modification (5.1); the second variant includes both introduced modifications (5.1 and 5.2). The comparison of running times and results of these algorithms show the benefit of the altered algorithm.

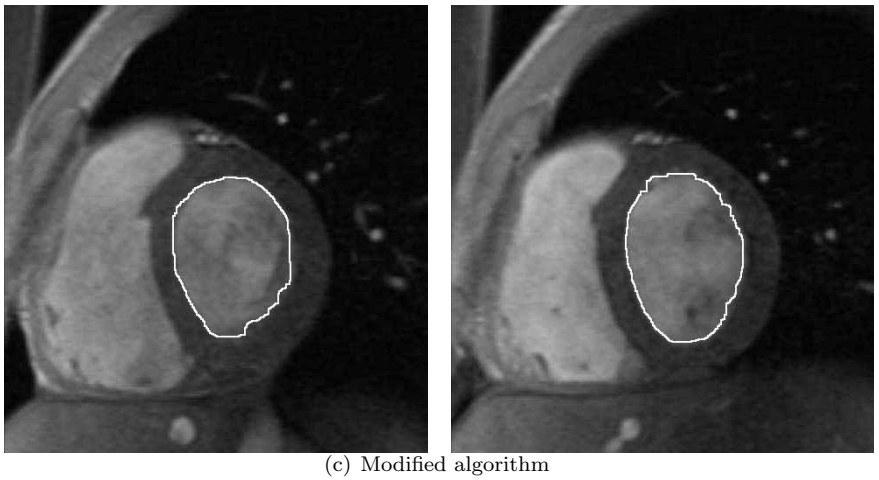
Figure 6.1 (page 53) demonstrates the results of both implementations of the Edmonds–Karp algorithm. The resolution of the pictures is 484×512 px. Black lines in the initial images 6.1(a) denote object seeds and white lines represent background seeds. Segmentation results in 6.1(b) and 6.1(c) are indicated by white borders. Same parameter $\lambda = 2$ was used in all cases. The differences between the results of these two algorithms are significant. The original Edmonds–Karp algorithm marks many areas of the image as an object while the modified algorithm segments only one area as intended. Notice that the common areas of the results of both algorithms are identical.



(a) Initial images



(b) Edmonds–Karp algorithm



(c) Modified algorithm

FIG. 6.1. *Comparison of the results of both implementations.*

Table 6.1 summarizes the running times of the Edmonds–Karp algorithm (E–K) and the modified algorithm (MOD) processing three sample images in three resolutions. All tests were performed on a CPU Intel Core 2 Duo P8400 (2.26 GHz), program only made use of one CPU core. The values show that the presented modification 5.2 speeds-up the segmentation process roughly five to ten times depending on the initial data.

| | **1** | | **2** | | **3** | |
|---|---|---|---|---|---|---|
| **Resolution** | E–K | MOD | E–K | MOD | E–K | MOD |
| 121×128 px | 0.182 s | 0.024 s | 0.220 s | 0.015 s | 0.188 s | 0.020 s |
| 242×256 px | 0.971 s | 0.173 s | 1.334 s | 0.147 s | 1.211 s | 0.186 s |
| 484×512 px | 14.407 s | 2.795 s | 14.443 s | 1.860 s | 15.592 s | 2.837 s |

TABLE 6.1
*Running times of both implementations.*

**7. Conclusion.** In this paper we present the basic graph cut approach to the image segmentation. We introduce our modifications that improve the basic Edmonds–Karp algorithm. Running times are significantly shorter and the resulting segmentation of an image corresponds to the expected result more precisely.

**Acknowledgment.** The work was partly supported by the project No. MSM 6840770010 "Applied Mathematics in Technical and Physical Sciences", by the project No. LC06052 "Jindřich Nečas Center for Mathematical Modelling", both of the Ministry of Education, Youth and Sports of the Czech Republic, and by the project "Advanced Supercomputing Methods for Implementation of Mathematical Models" of the Student Grant Agency of the Czech Technical University in Prague No. SGS11/161/OHK4/3T/14.

REFERENCES

[1] Y. Boykov, G. Funka-Lea. *Graph Cuts and Efficient N-D Image Segmentation.* International Journal of Computer Vision **70**, Issue 2 (2006), 109–131.

[2] Y. Boykov, M.-P. Jolly. *Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images. Proceedings of "International Conference on Computer Vision".* 2001, vol.1, p. 105-112.

[3] Y. Boykov, 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**, Issue 9 (2004), 1124–1137.

[4] Y. Boykov, O. Veksler. *Graph Cuts in Vision and Graphics: Theories and Applications.* in Handbook of mathematical models in computer vision. New York: Springer, 2006, p. 605.

[5] L. R. Ford, Jr., D. R. Fulkerson. *Maximal flow through a network.* Canadian Journal of Mathematics **8** (1956), 399–404.

[6] D. Freedman, T. Zhang. *Interactive Graph Cut Based Segmentation With Shape Priors.* Computer Vision and Pattern Recognition **1** (2005), 755–762.

[7] A. V. Goldberg, R. E. Tarjan. *A New Approach to the Maximum-Flow Problem.* Journal of the Association for Computing Machinery **35**, Vol. 35, No. 4, (1988), 921–940.

[8] D. Greig, B. Porteous, and A. Seheult. *Exact maximum a posteriori estimation for binary images.* Journal of the Royal Statistical Society, Series B **51**, No. 2 (1989) 271–279.

[9] M. Sonka, V. Hlavac, and R. Boyle. *Graph cut segmentation.* in Image processing, analysis, and machine vision. 3rd ed. Toronto: Thompson Learning, 2008, p. 298–306.

[10] S. Vicente, V. Kolmogorov, and C. Rother. *Graph cut based image segmentation with connectivity priors.* Computer Vision and Pattern Recognition 2008, pp. 1-8.