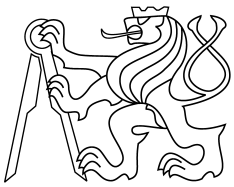
CENTER FOR
MACHINE PERCEPTION



CZECH TECHNICAL
UNIVERSITY IN PRAGUE

MASTER'S THESIS

ISSN 1213-2365

Discrete Energy Minimization with Global Constraints

Valerii Ulitin

ulitival@fel.cvut.cz

May 24, 2017

Thesis Advisor: RNDr. Daniel Průša, Ph.D.

Research Reports of CMP, Czech Technical University in Prague,

Published by

Center for Machine Perception, Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University
Technická 2, 166 27 Prague 6, Czech Republic
fax +420 2 2435 7385, phone +420 2 2435 7637, www: http://cmp.felk.cvut.cz

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

DIPLOMA THESIS ASSIGNMENT

Student: Valerii U l i t i n

Study programme: Open Informatics

Specialisation: Computer Vision and Image Processing

Title of Diploma Thesis: Discrete Energy Minimization with Global Constraints

Guidelines:

Discrete energy minimization (a.k.a. MAP inference in graphical models) is a well known and intensively studied optimization problem which has many applications in low level computer vision. The goal of the thesis is to formulate extensions of the problem by introducing additional global constraints and propose algorithms for solving them. The global constraints should help e.g. to perform the image segmentation when a sufficient quality model of the foreground and background is not available, on the other hand there is some information on cardinalities or number of components of particular segmentation groups.

- Summarize related state of the art methods working with global constraints ([4] - section 6.3, [5]).
- Formulate your own extensions (based on experiments carried out within A4M33SVP) for a) submodular instances with binary variables [1] and b) metric instances [2].
- Describe complexity of the extended problems and propose fast (approximation) algorithms to solve them.
- Evaluate the performance of the algorithms. Apply them to the image segmentation task. Compare the achieved results with the results of standard methods (not working with global constraints).

Bibliography/Sources:

[1] Y. Boykov, V. Kolmogorov: An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision, IEEE Trans. on Pattern Recognition and Machine Intelligence, 2004.
[2] Y. Boykov, O. Veksler, R. Zabih: Fast Approximate Energy Minimization via Graph Cuts, IEEE Trans. on Pattern Recognition and Machine Intelligence, 2001.
[3] B. W. Kernighan, S. Lin: An Efficient Heuristic Procedure for Partitioning Graphs, Bell Labs Technical Journal, Volume 49, Issue 2, 1970.
[4] T. Werner: Revisiting the Linear Programming Relaxation Approach to Gibbs Energy Minimization and Weighted Constraint Satisfaction, IEEE Trans. on Pattern Recognition and Machine Intelligence, 2010.
[5] A. Delong, L. Gorelick, O. Veksler, Y. Boykov: Minimizing Energies with Hierarchical Costs, International Journal of Computer Vision, Volume 100, Issue 1, 2012.

Diploma Thesis Supervisor: RNDr. Daniel Průša, Ph.D.

Valid until: the end of the summer semester of academic year 2017/2018

L.S.

prof. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, December 21, 2016

Acknowledgment

I would like to thank all the people who helped me with my diploma thesis in any way. I thank my thesis supervisor RNDr. Daniel Průša, Ph.D. for valuable comments and remarks he had given me during the creation of this thesis. My thanks also goes to my wife and my family for supporting and encouraging me.

Author's declaration

I declare that I have work out the presented thesis independently and that I have listed all information sources used in accordance with the Methodical Guidelines about Maintaining Ethical Principles for Writing Academic Theses.

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne ....

Podpis autora práce

Abstract

The results of this work are algorithms that allow a segmentation of the sequence of similar images, where user annotates just one image from the sequence. These algorithms can be useful while dealing with large amount of data. Created solutions provide the ability to segment similar scenes based on the one generated color model. The goal of this work is to find algorithms that would effectively solve the problem of segmentation of the image sequences of one scene, where the color model is generated only for few images. A knowledge of the foreground pixels number is assumed. Two algorithms suitable for this task are presented. The first algorithm uses the bisection method to find the most optimal value Δ, which will then be subtracted from the unary weights of each pixel. This algorithm can work with multiple labels by using the *alpha expansion* method. The second algorithm performs so-called pixel swapping using heaps, where it tries to reduce overall segmentation energy while fulfilling the global constraints and it is primarily used only when the first one fails.

Keywords: Energy minimization, graph algorithms, minimum cut, maximum flow, image segmentation

Abstrakt

Výsledkem této práce jsou algoritmy umožňující segmentaci sekvence podobných obrázků, kdy uživatel anotuje pouze jeden obrázek ze sekvence. Takové algoritmy můžou být přínosné při zpracování velkého množství dat. Vytvořená řešení poskytují možnost na základě jednoho vygenerovaného barevného modelu úspěšně segmentovat podobné scény. Cílem této práce je návrh algoritmů, které by efektivně řešily problém segmentace velké množiny podobných obrázků. Práce prezentuje dva takovéto algoritmy, jež dokážou zlepšit (minimalizovat) celkovou energii při provedení segmentace na sekvenci obrázků. První algoritmus využívá metody půlení intervalu při hledání nejoptimálnější hodnoty Δ, jež je pak odečtena od unárních vah každého pixelu. Tento algoritmus dokáže pracovat i s více značkami, a to tak, že využívá metodu *alfa expanze*. Druhý algoritmus provádí tzv. prohození pixelů s využitím hald, kde v každém prohazování se snaží zmenšit celkovou energii segmentace a je primárně používán pouze v případě selhání prvního algoritmu.

Klíčová slova: Minimalizace energie, grafové algoritmy, minimální řez, maximální tok, segmentace obrázků

Contents

List of Figures

List of Tables

List of Algorithms

List of Symbols and Abbreviations

HOP	High order potentials.
MAP	Maximum a posteriori.
MRF	Markov random fields.
GMM	Gaussian mixture model.
3D	Three dimension(al).
RGB	Red, green, blue.
GB	Gigabyte.
OS	Operating system.
JDK	Java development kit.
CPU	Central processing unit.

1. Introduction

Energy minimization is a well known problem which has a strong practical and theoretical importance for computer vision [1]. In general energy can express how solution of a computer vision task is good or bad so far. High energy leads to a bad result whereas low energy leads to a good one. It is known, that even for low-level vision tasks solving energy minimization problem can be hard (often NP-hard).

In this work we designed two algorithms that are capable of minimizing energy in the task of image sequences segmentation by a given model. We are not improving existing algorithms for achieving better segmentation results, but we are designing algorithms that can effectively solve one particular problem which is energy minimization with global constraints, *e.g.* segmentation a set of similar images using color model obtained only from one annotated image from this set. Those algorithms work in polynomial time using some heuristic approach and knowledge about the domain.

It is known that energy minimization problem can be NP-hard, *e.g.* when not dealing with binary labels, and introducing another global constraints only make this problem more difficult.

For our algorithm as an initial step we used output of max-flow algorithm [10], then we proceed in two ways solving the energy minimization task with global constraints:

1. perform Δ-max-flow, described in Section 4,

2. perform pixels swapping, described in Section 5.

Both algorithms repeat until they met desired results on segmented objects size. Here, the second algorithm is used in cases where Δ-max-flow failed to converge to desired results and the output of Δ-max-flow is further used in the *pixel swapping* algorithm as an initial solution.

1.1. Motivation

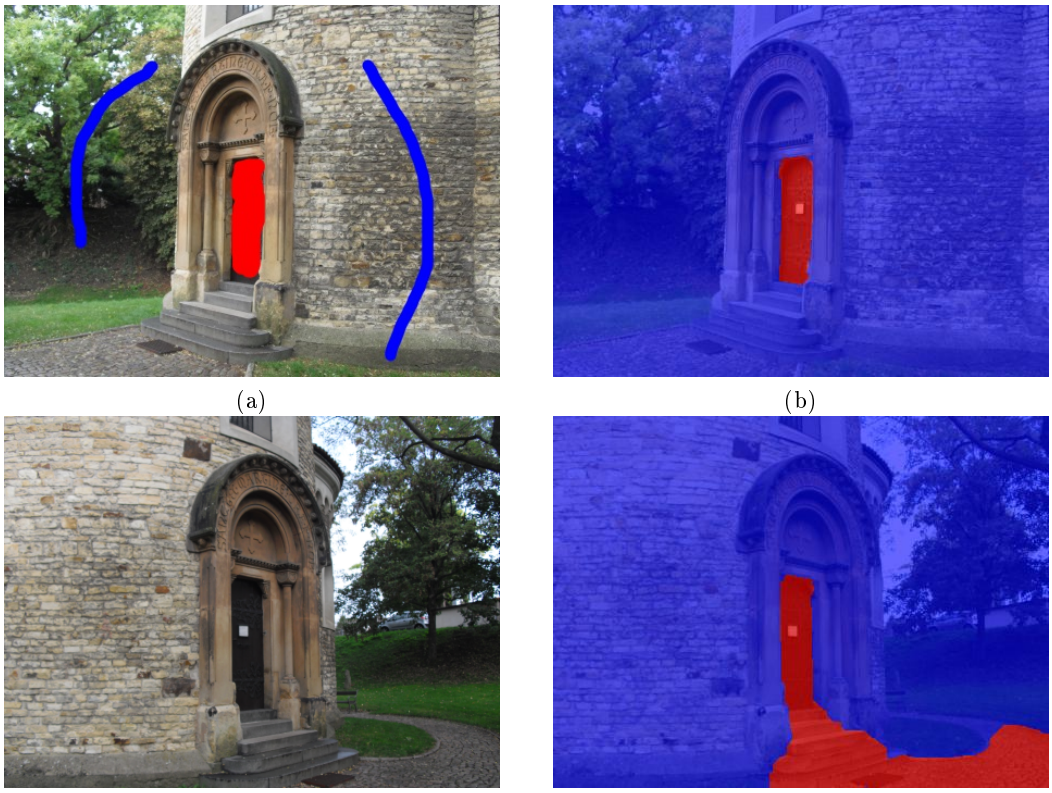
Image segmentation is a widely known problem in computer vision. It stands for detecting an object of interest and split it from the background or other objects. Sonka et al. [2] wrote following about segmentation:

> *Image segmentation is one of the most important steps leading to the analysis of processed image data – its main goal is to divide an image into parts that have a strong correlation with objects or areas of the real world contained in the image. We may aim for* ***complete segmentation****, which results in a set of disjoint regions corresponding uniquely with objects in the input image, or for* ***partial segmentation****, in which regions do not correspond directly with image objects. To achieve a complete segmentation, cooperation with higher processing levels which use specific knowledge of the problem domain is necessary. ([2], page 175)*

We are interested in segmentation in the case of a global context, because our goal is to design such an algorithm, which can perform segmentation based on some knowledge about the scene. There is a problem of object retrieval from the image sequences. It is a

complicated task to obtain desired objects from the whole sequence of images in terms of time. Generally, to successfully perform segmentation on image, we need to manually annotate a sample of our desired object (foreground) and a sample of what does not matter (background). In the case when there are 1000 images of the same scene and we need to retrieve some object from the whole set one can imagine what time it will take. So, what are the alternatives?

Theoretically if the scene does not change much one can obtain a color model (2.2) of the first image and apply it to the rest of the set, but this solution can, and probably will, lead to errors in segmentation (e.g. there will not be the desired object, something else would be segmented or we segment our target together with not relevant one). Figure 1.1 shows an example of using an acquired model on the images of the same scene.



(a) (b) (c) (d)

Figure 1.1. Annotated image **(a)** is used for obtaining a color model. In image **(b)** one can see segmentation performed on image **(a)**. Image (c) shows the image on which we attemp to apply the trained model. In image **(d)**, a segmentation error can be easily seen after the model from image **(a)** was applied.

The idea of re-using the trained model is not bad, but requires some tweaking, hence we introduce global constraints for the task of image sequences segmentation. A suitable example of such constraints can be the object size - Figure 1.2 shows correct segmentation of 1.1(c) after we added this size constraint.

The size constraint can be described as an interval between two numbers:

$$sz = [a, b] \quad a, b \in \mathbb{R} \tag{1.1}$$

In this work we study algorithms that can, based on a fixed color model obtained from one image, perform segmentation on an images sequence in relatively short time without the user's interaction. It means that by a given model we will acquire object

Figure 1.2. Correct segmentation of 1.1(c) after applying a size constraint to the task.

of interest from all images within the sequence. This would probably save a lot of time spent on manual processing.

1.2. State of the art review

There are several different approaches for solving the problem of energy minimization with global constraints for graphs. Gupta et al. [6] designed fast inference algorithm for graphical models where the clique potentials comprise of two parts: the first part is sum of individual node potentials (unary energy) and, the second part is a symmetric n-ary function that depends only on the number of nodes that get a particular label. In their work these clique potentials are called *cardinality-based*.

Next approach was made by Tarlow et al. [8]. Their goal was to be able to use a broad range of high order potentials (HOPs) generically within MAP message passing algorithms as easily as used low order tabular potentials. They defined three issues that must be addressed:

1. Message updates need to be computed effectively even when factors range over very large subsets.

2. It should be easy to recognize when problems contain tractable high order structure.

3. HOP constructions should be flexible and reusable, not requiring new problem-specific derivations and implementations on each use.

Werner in [9] showed that is possible to handle high-arity MRF with global constraints. He described them as an *extensional* representation which means that it can be represented by explicitly or as an *intensional* representation which means that constraint is a black-box function. Intensionally represented constraints of a non-fixed arity are referred to *global constraints*.

He designed algorithms that are capable of solving such high-arity problems with global constraints using plausible approximation factor.

Another solution that also uses global constraints but has different approach than ones described above was introduced by Delong et al. [11]. Their work is based on using ‘context‘ to resolve ambiguities in object recognition. The main idea is that certain groups of labels are self-consistent because they tend to appear together, *e.g.* the

{*car, road, sky*} all belongs to "outdoors" context. Based on this idea they introduced *hierarchical costs* for labels that are explicitly grouped in a hierarchy. Their algorithm *h-fusion* therefore solves optimization problem of hierarchical costs minimization.

These mentioned above approaches use message passing for solving problems with global constraints and it is known that message passing is usually slower than combinatorial algorithms. Also message passing can not handle some of global constraints, *e.g.* constraint on number of components. When combinatorial approaches can handle such constraints.

Unlike them we took the different way in solving this task, seeing the problem as a combinatorial optimization and solving it in a similar manner as Kernighan and Lin described in their paper [7].

2. Prerequisites

2.1. Problem instance

Image segmentation is cast as the optimization problem of energy minimization in the form of:

$$E(x) = \theta_{const} + \sum_{p \in \mathcal{V}} \theta_p(x_p) + \sum_{(p,q) \in \mathcal{E}} \theta_{pq}(x_p, x_q). \tag{2.1}$$

Here, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is an undirected graph. Set $\mathcal{V}$ corresponds to pixels; x_p denotes the label of pixel $p \in \mathcal{V}$ which belongs to the set $\{0, 1, ..., L-1\}$, where L is the number of labels. In our case it is possible to ignore the constant term θ_{const}, hence we set $\theta_{const} = 0$. $\theta_p(x_p)$ denotes the unary terms which encode data penalty function, and $\theta_{pq}(x_p, x_q)$ represents the pairwise terms which are interaction potentials (edge costs in graph $\mathcal{G}$).

For unary terms we compute probabilities of label $i \in \{0, 1, ..., L-1\}$ assigned to pixel p using Gaussian Mixture Model (*GMM*)[2] for all labels.

$$p_i(x; \theta_i) = \sum_{k=1}^{K} \phi_{i,k} \, \mathcal{N}(x; \mu_{i,k}, \Sigma_{i,k}). \tag{2.2}$$

Here K is the number of components used in *GMM* estimation. Both foreground and background *GMM* is fully specified by a set of its parameters:

$$\theta_i = \{\phi_{i,1} ... \phi_{i,K}, \mu_{i,1} ... \mu_{i,K}, \Sigma_{i,1} ... \Sigma_{i,K}\}.$$

Here $\phi_{i,k}$ denotes the prior probability for k-th component of label i. It holds

$$\sum_{k=1}^{K} \phi_k = 1.$$

$\mu_{i,k}$ represents the mean value of k-th component of label i and $\Sigma_{i,k}$ denotes covariance matrix of k-th component of label i. $\mathcal{N}(x; \mu_{i,k}, \Sigma_{i,k})$ stands for K multivariate normal distribution.

In our segmentation task we represent an image as a graph in which each node has 8 neighbors except nodes on image borders, these have five neighbors, and nodes in corners, these have 3 neighbors, example of such a graph is shown in Figure 2.1.

Pairwise terms are obtained as edge costs in graph $\mathcal{G}$ using the following equation:

$$\theta_{pq}(x_p, x_q) = \lambda_1 + \frac{\lambda_2}{d(p,q)} e^{-\frac{\|x_p - x_q\|^2}{2\beta}}$$

Here λ_1 is the weight of Ising term in pairwise costs and λ_2 is the weight of dependent term. Term $d(p, q)$ is the Euclidean distance in image grid (neighboring pixels). It is equal to 1 for horizontally and vertically neighboring pixels and it is equal to $\sqrt{2}$ for diagonal neighbors. The term $\|x_p - x_q\|^2$ denotes squared Euclidean distance of pixel colors. The pixel color $x_p \in [0, 1]^3$ is a 3D vector of RGB components.

$$\|x_p - x_q\|^2 = (x_p^R - x_q^R)^2 + (x_p^G - x_q^G)^2 + (x_p^B - x_q^B)^2$$

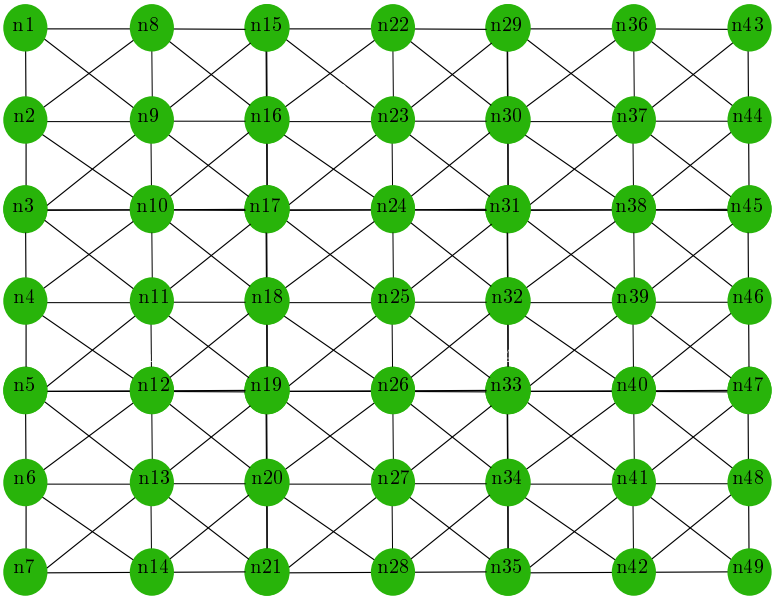


Figure 2.1. Graph representation of an image. Corner nodes have 3 neighbors, the other border nodes have 5 neighbors, remaining nodes have 8 neighbors. *E.g.*, n1 has {n2, n8, n9} set of neighbors, n15 has {n8, n9, n16, n23, n22}, n11 has {n3, n4, n5, n12, n19, n18, n17, n10}.

The term β normalizes Euclidean distance of pixel colors. It is equal to the expected squared Euclidean distance of pixel colors in image, i.e. over each pair of neighbors $\{p', q'\} \in \mathbb{N}$, where $\mathbb{N}$ is a set of all neighboring pixels in a graph $\mathcal{G}$:

$$\beta = \frac{1}{|\mathbb{N}|} \sum_{(p',q') \in \mathbb{N}} \left\| x_{p'} - x_{q'} \right\|^2$$

We will consider two cases in this work:

1. The possible labeling is binary, i.e. $x_p \in \{0, 1\}$
2. The metric instance of the problem, where the number of possible labeling can be arbitrary, i.e. $x_p \in \{0, 1, \ldots, L-1\}$

The case where we deal only with binary labeling can be reduced to max-flow problem. It is known that graph cuts algorithms can be used to find an optimal solution for *submodular* energy functions, i.e. functions whose pairwise terms satisfy:

$$\theta_{pq}(0,0) + \theta_{pq}(1,1) \leq \theta_{pq}(0,1) + \theta_{pq}(1,0).$$

This is a exact example of binary segmentation task described above.

In this work we call a problem instance whose energy terms meet these conditions as *submodular instance problem.*

In situation when the cardinality of the set of labels is ≥ 3, we use the expansion move algorithm (α – expansion) [3]. In this case energy function pairwise terms should satisfy:

$$\theta_{pq}(\beta,\gamma) + \theta_{pq}(\alpha,\alpha) \leq \theta_{pq}(\beta,\alpha) + \theta_{pq}(\alpha,\gamma), \tag{2.3}$$

which must hold for all labels $\alpha, \beta, \gamma \in \{0, \ldots, L-1\}$.

This condition says that penalty for changing labeling from β to γ while α is preserve its labels should be less or equal than penalty for changing labels from β to α and as

follows from α to γ.

Problems that meet conditions from (2.3) are called *metric instance problems*. They also include submodular instance for two labels.

2.2. Max-flow

In order to perform minimization of submodular two-label energy using max-flow algorithm we first mention the following notation introduced by Kolmogorov and Rother [1]. The energy of (2.1) is specified by the constant factor θ_{const}, unary terms $\theta_p(i)$, and pairwise terms $\theta_{pq}(i,j)$ where $(i,j) \in \{0,1\}$. The last two terms are denoted as $\theta_{p;i}$ and $\theta_{pq;ij}$, respectively.

Further these notations are simplified to θ_p to denote a vector of size 2 and θ_{pq} to denote a vector of size 4, but since it holds for our task, that $\theta_{pq;00} \equiv \theta_{pq;11}$ and $\theta_{pq;10} \equiv \theta_{pq;10}$ we can describe θ_{pq} as a scalar.

The energy (2.1) is therefore completely specified by parameter vector θ, where pa-

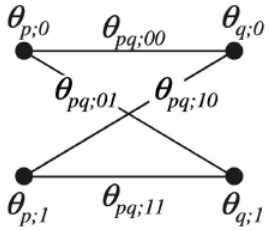


Figure 2.2. Convention for displaying parameters $\theta_p, \theta_{pq}, \theta_q$ [1].

rameters of the energy are shown in Figure 2.2.

Kolmogorov and Rother [1] describe in their work an algorithm for submodular functions. It consists of three steps:

1. convert the energy to a "Normal form".
2. construct a directed weighted graph $G = (V, A, c)$. Here, V is the set of vertices, A is the set of arcs and c is the capacity, s.t. $c : A \to R^+$. There are two special nodes - source s and sink t, called the *terminals*.
3. find minimum $s - t$ cut for constructed graph G.

Notion "Normal form" as described in [1] means that the vector θ is in a *normal form* if it satisfies the following:

1. $\min\{\theta_{p;0}, \theta_{p;1}\} = 0$ for all pixels p.
2. $\min\{\theta_{pq;0j}, \theta_{pq;1j}\} = 0$ for all directed edges $(p \to q)$ and labels $j \in \{0,1\}$.

So in the algorithm optimizing label assignment task for submodular function [1] the first step is the preprocessing which converts vector θ into the normal form. This can be done in two phases as follows:

1. While there is an edge $(p \to q)$ and a label j violating Condition 2, do the following:

   compute $\delta = \min\{\theta_{pq;0j}, \theta_{pq;1j}\}$;

   set $\theta_{pq;0j} := \theta_{pq;0j} - \delta$, $\theta_{pq;1j} := \theta_{pq;1j} - \delta$, $\theta_{q;j} := \theta_{q;j} + \delta$.

Table 2.1. Rules for adding an edge to the graph G

component of θ	corresponding edge $a \in A$	capacity c_a
$\theta_{p;0}$	$(p \rightarrow t)$	$\theta_{p;0}$
$\theta_{p;1}$	$(s \rightarrow p)$	$\theta_{p;1}$
$\theta_{pq;01}$	$(p \rightarrow q)$	$\theta_{pq;01}$
$\theta_{pq;10}$	$(q \rightarrow p)$	$\theta_{pq;10}$

2. For every pixel p:

   compute $\delta = \min\{\theta_{p;0}, \theta_{p;1}\}$

   set $\theta_{p;0} := \theta_{p;0} - \delta$, $\theta_{p;1} := \theta_{p;1} - \delta$, $\theta_{const} := \theta_{const} + \delta$.

It is seen that after converting energy to a normal form, unary and pairwise terms become submodular.

Algorithm for Submodular Functions

The following algorithm from [1] is used to assign labels to pixels in the case when all terms of energy (2.1) are submodular.

Algorithm 1: Algorithm for submodular functions

Data: θ unary and pairwise energy terms
Result: $x_p \in X$ with assigned labels
1. $\tilde{\theta}$ = normalize (θ);
2. create s, t;
3. create $G = (V, A, c)$ from $\tilde{\theta}$;
4. add $\{s, t\}$ to V;
5. $X = maxflow(G)$;
6. Return X

Here V corresponds to pixels set $\mathcal{V}$ and s, t are the terminals. For every nonzero component of θ, an edge is added to A according to the rules described in Table 2.1

The process of constructing the directed graph from energy terms is shown in Figure 2.3. After constructing the graph we get the minimum $s - t$ cut (S,T) by computing

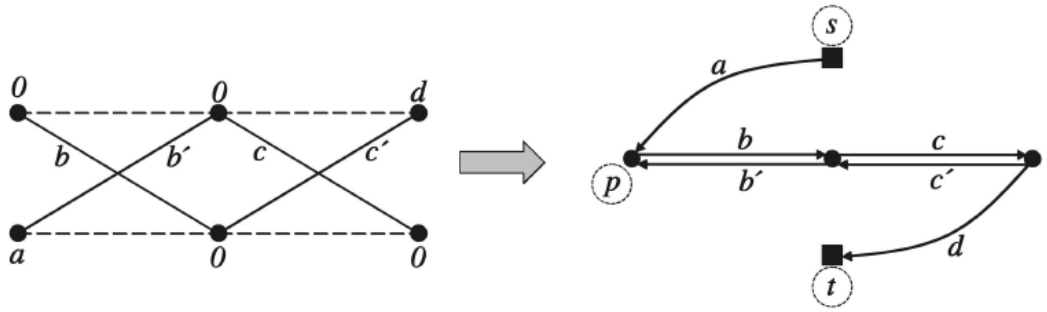


Figure 2.3. Graph construction. Left: input energy function (in a normal form) with three nodes and two edges. Right: corresponding graph G [1].

maximum flow from the source to the sink. This cut defines labeling X as follows:

$$x_p = \begin{cases} 0 & \text{if} \quad p \in S \\ 1 & \text{if} \quad p \in T \end{cases}$$

It can be seen that the cost of any cut is equal to the energy of the corresponding labeling plus the constant term θ_{const}.

2.3. α – expansion

For solving segmentation task where cardinality of labels set is greater or equal three it is not possible to use submodular maximum flow directly. Let us introduce the *metric* notation of pairwise penalty V. Boykov et al. [3] in their work call V a *metric* on the space of labels L if it satisfies

$$V(\alpha, \beta) = 0 \quad \Leftrightarrow \quad \alpha = \beta, \tag{2.4}$$

$$V(\alpha, \beta) \quad = \quad V(\beta, \alpha) \geq 0, \tag{2.5}$$

$$V(\alpha, \beta) \quad \leq \quad V(\alpha, \gamma) + V(\gamma, \beta), \tag{2.6}$$

for any labels $\alpha, \beta, \gamma \in L$. If V satisfies only (2.3) and (2.4), it is called a *semimetric*.

For expansion algorithm we are interested only in *metric* penalization. Algorithm of α – expansion that authors described in their work [3] is shown in 2.

Algorithm 2: α – expansion algorithm

Data: arbitrary labeling f
Result: labeling f that minimizes an overall energy $E(f)$
1. *success* := 0;
2. **for** *each label* $\alpha \in L$ **do**
   2.1 Find $\hat{f} = argmin\, E(f')$ among f' within one α – expansion of f;
   2.2 If $E(\hat{f}) < E(f)$, set $f := \hat{f}$ and success := 1;
end
3. If success = 1 goto 1;
4. Return f;

The main idea of α – expansion algorithm is to compute minimum cut on a graph $G_\alpha = \langle \mathcal{V}_\alpha, \mathcal{E}_\alpha \rangle$. The structure of this graph is determined by the current partition **P** (assigned labels to pixels) and by the label α. A representation of graph G is illustrated in Figure 2.4.

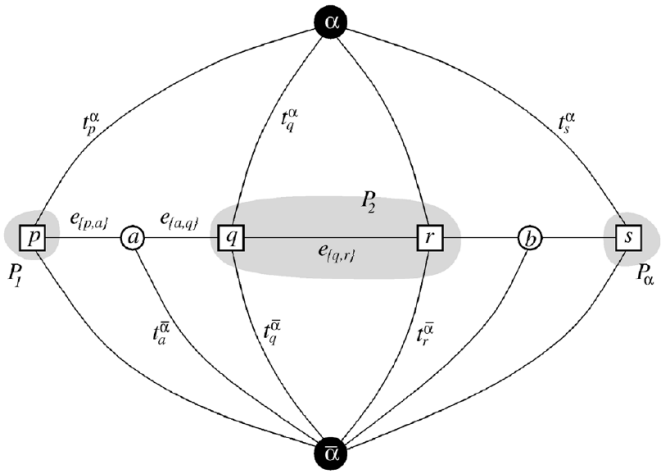


Figure 2.4. An example of graph G_α for a 1D image. The set of pixels in the image is $\mathcal{P} = \{p, q, r, s\}$ and the current partition is $\mathbf{P} = \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_\alpha\}$, where $\mathcal{P}_1 = \{p\}$, $\mathcal{P}_2 = \{q, r\}$, and $\mathcal{P}_\alpha = \{s\}$. Two auxiliary nodes $a = a_{p,q}$, $b = a_{r,s}$ are introduced between neighboring pixels separated in the current partition. Auxiliary nodes are added at the boundary of sets $\mathcal{P}_l$ [3].

The set of nodes $\mathcal{V}$ includes the two terminals α and $\bar{\alpha}$, as well as all image pixels $p \in \mathcal{P}$, where $\mathcal{P}$ stands for a set of pixels in image. In addition, for each pair of neighboring pixels $\{p, q\} \in \mathbb{N}$ separated in the current partition (i.e., such that $f_p \neq f_q$, where f is the current labeling), *auxiliary node* a_{pq} is created. Auxiliary nodes are introduced at the boundaries between partition set $\mathcal{P}_l$ for $l \in \mathcal{L}$. Thus, the set of all nodes is:

$$\mathcal{V}_\alpha = \left\{ \alpha, \bar{\alpha}, \mathcal{P}, \bigcup_{\substack{\{p,q\}\in\mathbb{N} \\ f_p \neq f_q}} a_{pq} \right\}.$$

Each pixel $p \in \mathcal{P}$ is connected to the terminals α and $\bar{\alpha}$ by t-links t_p^α and $t_p^{\bar{\alpha}}$, respectively. For each pair of neighboring pixels $\{p, q\} \in \mathbb{N}$ such that $f_p \neq f_q$, we create a triplet of edges $\mathcal{E}_{p,q} = \{e_{p,a}, e_{a,q}, t_p^{\bar{\alpha}}\}$, where $a = a_{pq}$ is corresponding auxiliary node. The edges $e_{p,a}$ and $e_{a,q}$ connect pixels p and q to $a_{p,q}$ and the t-link $t_p^{\bar{\alpha}}$ connects the auxiliary node a_{pq} to the terminal $\bar{\alpha}$. So, the set of all edges can be written as:

$$\mathcal{E}_\alpha = \left\{ \bigcup_{p\in\mathcal{P}} \{t_p^\alpha, t_p^{\bar{\alpha}}\}, \bigcup_{\substack{\{p,q\}\in\mathbb{N} \\ f_p \neq f_q}} \mathcal{E}_{p,q}, \bigcup_{\substack{\{p,q\}\in\mathbb{N} \\ f_p = f_q}} e_{p,q} \right\}.$$

The edges weights are assigned according to Table 2.2.

After getting the minimum cut C on the graph $\mathcal{G}_\alpha$ we get the labeling f^C corresponding to this cut. Formally,

$$f_p^C = \begin{cases} \alpha & if \quad t_p^\alpha \in C \\ f_p & if \quad t_p^{\bar{\alpha}} \in C \end{cases} \quad \forall p \in \mathcal{P}.$$

In other words, as it is described in [3], a pixel p is assigned label α if the cut C separates p from the terminal α, while p is assigned its old label f_p if C separates p from $\bar{\alpha}$. Note that, for $p \notin \mathcal{P}_\alpha$, the terminal $\bar{\alpha}$ represents labels assigned to pixels in the initial labeling f. Boykov et al. [3] stated the following lemma:

Table 2.2. Weights assigned to the edges in graph $\mathcal{G}$

edge	weight	for
$t_p^{\bar{\alpha}}$	∞	$p \in \mathcal{P}_\alpha$
$t_p^{\bar{\alpha}}$	$\theta_p(f_p)$	$p \notin \mathcal{P}_\alpha$
t_p^{α}	$\theta_p(\alpha)$	$p \in \mathcal{P}$
$e_{p,a}$	$\theta_{pq}(f_p, \alpha)$	$\{p,q\} \in \mathbb{N}, f_p \neq f_q$
$e_{a,q}$	$\theta_{pq}(\alpha, f_q)$	
$t_p^{\bar{\alpha}}$	$\theta_{pq}(f_p, f_q)$	
$e_{p,q}$	$\theta_{pq}(f_p, \alpha)$	$\{p,q\} \in \mathbb{N}, f_p = f_q$

Lemma 2.3.1 *A labeling f^C corresponding to a cut C on $\mathcal{G}_\alpha$ is one α – expansion away from the initial labeling f.*

Both algorithms, max-flow for submodular functions and α – expansion, will be actively used further in this work.

3. Δ-max-flow

In order to be capable of solving segmentation tasks for the image sequences we expand the algorithm of maximum flow for *submodular instance problem* and the α – expansion algorithm for *metric instance problem* to cope with the global constraints.

Submodular instance problem

As we could see in the Introduction part we can not just create a color model from the first image in the images set and apply it to the rest, because in this case we will end up with many errors and inaccuracies. Let us introduce an *object size* as a new global constraint to this problem, so we can say that our max-flow based algorithm performs correct segmentation if the searched object has size that lies in a defined interval.

To handle this global constraint we introduce Δ which is the new term for unary vector. We are going to re-balance unary weights by adding or subtracting Δ from $\theta_{p;1}$ until we met desired results in object size. By using Δ we can achieve more accurate results, because after adding or subtracting and perform new max-flow cardinality of current target labeling f can change. We only need to find such a Δ that will lead us to correct interval sz (1.1).

So the new unary term will be computed in the following manner:

$$\theta^{\Delta}_{p;1} = \theta_{p;1} \pm \Delta, \tag{3.1}$$

here, $\pm$ means that in case when we are interested to make foreground label ($x_p = 1$) more favorable we then decrease cost of $\theta_{p;1}$ by Δ, the other way around when we are interested to place foreground label at a disadvantage we then increase cost of $\theta_{p;1}$ by Δ.

Lets us modify previously defined max-flow algorithm 1 by adding Δ term. The algorithm is shown in 3.

Algorithm 3: Δ-max-flow algorithm

Data: θ unary and pairwise energy terms
Result: $x_p \in X$ with assigned labels
1. compute Δ;
2. $\theta^{\Delta}_{p;1}$ = update unary term $\theta_{p;1}$ by computed Δ;
3. $\tilde{\theta}$ = normalize (θ with $\theta^{\Delta}_{p;1}$);
4. create s, t;
5. create $G = (V, A, c)$ from $\tilde{\theta}$;
6. add $\{s, t\}$ to V;
7. $X = maxflow(G)$;
8. Return X

First of all we need to find Δ. Algorithm of finding Δ is shown in 4.

Algorithm 4: Algorithm computing Δ

Data: θ unary and pairwise energy terms
$[a, b]$ requested interval for object size, Section 2, (2.2)
Result: $x_p \in X$ with assigned labels
1. i = $\underset{i}{\text{argmax}}(\|\theta_{i;0} - \theta_{i;1}\|^2), \quad i = 1, \ldots, |\mathcal{P}|$;
2. Δ = compute interval bisection for interval $[0, |\theta_{i;0}| + |\theta_{i;1}|]$;
8. Return Δ

The algorithm 4 works iteratively until meets desired requests in interval size sz, where $sz = [a', b']$. Δ is computed for a foreground label using bisection interval method. It means that at the start we have such an interval $[a, b]$ that after adding $c = \frac{a+b}{2}$ to foreground unary terms and perform max-flow we will get segmentation where cardinality f of foreground pixels is the same as cardinality of all pixels in image.

In the next step we set a new interval $[a, b]$, s.t. $a = c$ if previous max-flow result led to cardinality of $f < a'$ or $b = c$ if previous max-flow result led to cardinality of $f > b'$.

Then we continue with a new interval $[a, b]$ until after adding or subtracting c we get cardinality of $f \geq a'$ and $f \leq b'$. After the algorithm ends the value c is our Δ.

After computing Δ we are going to update unary term using (3.1). All the next moves are classical max-flow which has already been described in Section 2.

This Δ method will help us achieve desired results on object size constraint, but there can be cases when this will not work. It means that algorithm after adding or subtracting even small Δ can not get into required interval. Example of this situation is shown in Figure 3.1, where we applied model acquired from image (a) of Figure 1.1 to image (c) of Figure 1.1.

Figure 3.4 illustrates a successful example of Δ-max-flow. For more illustrative results,

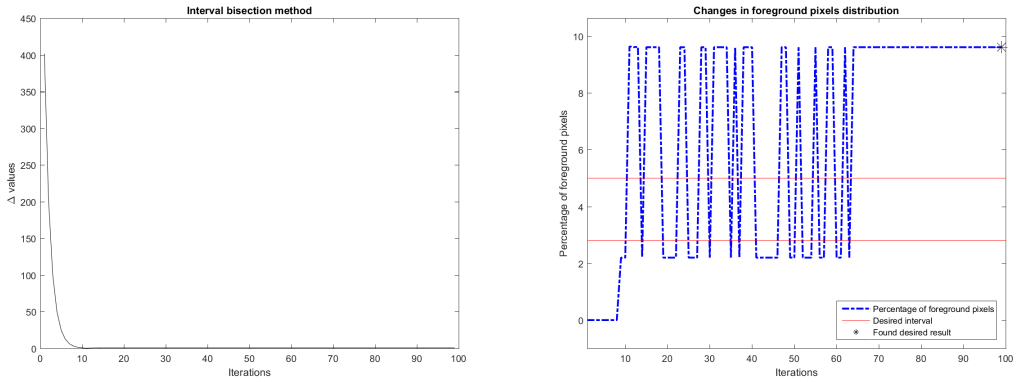


Figure 3.1. Example where Δ-max-flow will converge but not into desired interval. Left image shows changes in Δ. Right image shows percentage of foreground pixels and required interval, in this particular case it was set to [2.2, 5]. As it can be seen Δ-max-flow can not get into interval and after 65 iterations stacked in a wrong value. Algorithm was stopped after 100 iterations.

another image from the set was taken. This image is shown in Figure 3.2. Just as in previous example we applied the training model from the image (a) Figure 1.1. Results of segmentation after applying trained model and after Δ-max-flow are shown in Figure 3.3.

Figure 3.2. The image of Rotunda of St. Martin, Vyšehrad

(a) (b)

Figure 3.3. The image (a) demonstrates the result of segmentation just after applying trained model from the image (a) in Figure 1.1. Image (b) demonstrates results after applying Δ-max-flow.

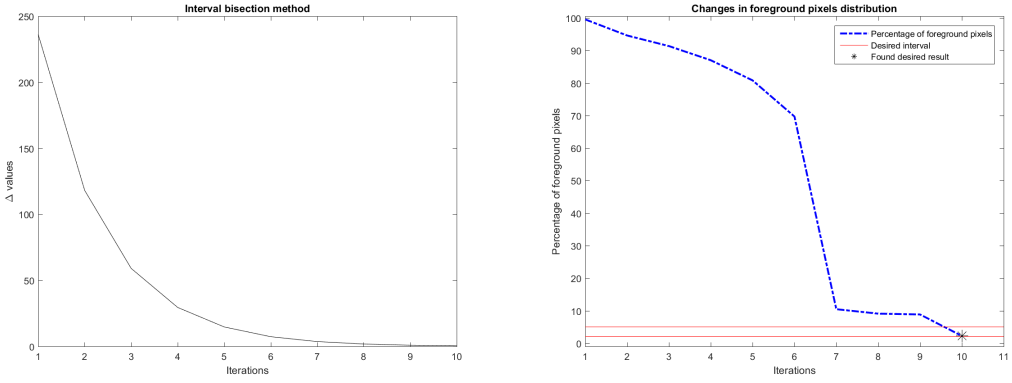


Figure 3.4. Example where Δ-max-flow successfully converged. The left image shows changes in Δ. The right image shows the percentage of foreground pixels and the required interval. Interval was set to [2, 5].

So, to handle bad algorithms outcome and achieved requested results we need to develop other method which can improve results of Δ-max-flow. For this particular situations we designed *pixel swapping algorithm* described in Section 4.

Metric instance problem

Further we also want to expand functionality of α – expansion algorithm using Δ approach to handle situations where we know about objects we are looking for and their sizes. Now we have different Δ for each possible α, so we will write it as Δ_α.

Just as in the case of the classical max-flow, we modify unary terms of the energy function (2.1). In α – expansion algorithm we iteratively add/subtract Δ from the current $\theta_{p;\alpha}$ using (3.1), so this operation looks like the following:

$$\theta^{\Delta}_{p;\alpha} = \theta_{p;\alpha} \pm \Delta_\alpha. \tag{3.2}$$

We then improve the algorithm of α – expansion 2 described in Section 2. Improved algorithm is given below 5.

Algorithm 5: Δ-α – expansion algorithm

Data: arbitrary labeling f
set $\mathcal{I}$ of objects size intervals

Result: labeling f that minimizes an overall energy $E(f)$ and all objects fits in their given intervals

1. $success := 0$;
2. **for** *each label* $\alpha \in L$ **do**
   - 2.1 Δ_α = compute Δ for current α, s.t. $|f_\alpha|$ fits into $\mathcal{I}_\alpha$;
   - 2.2 modify unary term $\theta_{p;\alpha}$ by Δ_α, $\forall p \in \mathcal{P}$;
   - 2.3 Find $\hat{f} = argmin\, E(f')$ among f' within one α – expansion of f;
   - 2.4 If $E(\hat{f}) < E(f)$, set $f := \hat{f}$ and success := 1;

   end
3. If success = 1 or not all objects fits into their intervals goto 1;
4. Return f;

Algorithm for computing Δ stays the nearly same. The only difference is that we are now looking for maximum Euclidean distance over all labels. In this case line 1 from algorithm 4 will have the following form:

$$i = \underset{i}{\operatorname{argmax}}(\left\|\theta_{i;\alpha} - \max(\theta_{i;\mathcal{L}\setminus\alpha})\right\|^2), \quad i = 1, \ldots, |\mathcal{P}|.$$

Here, $\mathcal{L}$ is the finite set of all possible labels that can occur in the current task. We are looking of a such index i, that maximizes the difference between the unary term θ_α and the maximum value from unary terms $\theta_{\mathcal{L}\setminus\alpha}$.

In the next step we calculate Δ_α using interval bisection optimization method. Inside this method we iteratively re-compute current Δ_α, modify unary terms and perform α – expansion algorithm with adjusted parameters. It repeats until we get such a Δ_α that will lead us to a solution where $|f_\alpha|$ lies inside the requested interval.

After obtaining Δ_α we once again modify the unary term θ_α, construct a graph $\mathcal{G}_\alpha$ in the same way as it is described in the Section 2 and compute the energy. These processes will repeat until none of the α will decrease the overall energy and cardinalities of all labels will not be in required intervals.

Process of the computing Δ_α is similar as in the case of Δ-max-flow describe above. The only difference is that we need to maintain constraints on object size at the same time. This means, that is not enough just to compute Δ_α for an one object save this Δ_α and move to another one. We need to iteratively change these Δ_αs in order to hold all constraints. It can happen that after we found Δ_α for one object and modify unary terms, another object which has been already processed in previous step can alter its size and we need to find new Δ_α for it. Figure 3.5 shows iteration process of fitting to desired object sizes for the image demonstrated in Figure 3.8.

Here, global constraint sz is written as a set of individual constraints for the each label in the form of:

$$sz = \{[a_1, b_1], [a_2, b_2], \ldots, [a_L, b_L]\}$$

where L is a number of possible labels.

Figure 3.8 illustrates an example of Δ-α – expansion. We take the first image from

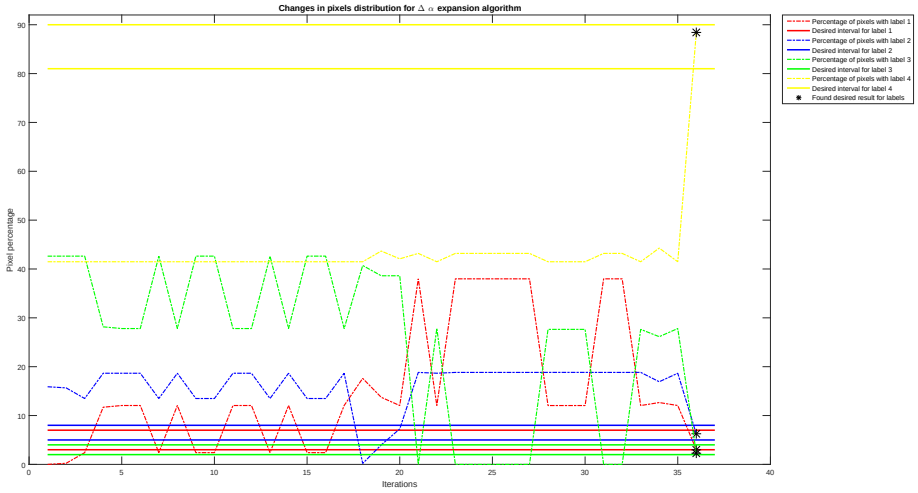


Figure 3.5. The figure demonstrates fitting process of labeling cardinality f during the Δ α – expansion algorithm running on image shown in Figure 3.7 (a). Here, desired intervals (percents) are: [3, 7] for the mole, [5, 8] for the penguin, [2, 4] for the mouse. Background is calculated as a rest, i.e. we subtract from 100 sum of all interval starting values a (from none background labels) and write it as b for background interval and the for a we subtract sum of all interval ending values b (from none background labels). We get the interval [81, 90] for the background.

the testing set for Δ-α – expansion algorithm. It is shown in Figure 3.6 and trained our model on this image. Next we applied gained model to the next image from the set, results are shown in Figure 3.7. This image is from the same scene but differs in a scale and an angle of view.

As it can be seen in Figure 3.7 after applying just simple model transfer we get horrible results, but if we afterwards apply Δ-α – expansion then results are improved dramatically, see Figure 3.8.

Figure 3.6. The first image of the testing set for the Δ-α – expansion algorithm with marked objects.

(a) (b)

Figure 3.7. (a) shows the testing image from the set on which we applied model gained from image illustrated in Figure 3.6. (b) shows output of pure α – expansion algorithm based on trained model.

Unfortunately just as in the case of *submodular instance problem* there can not exist guaranteed solution for every case. Example of the not so clear output which needs some additional enhancements is shown in Figure 3.9. It may be useful to use other relaxations, *e.g.* number of components constrain, for solving such cases.

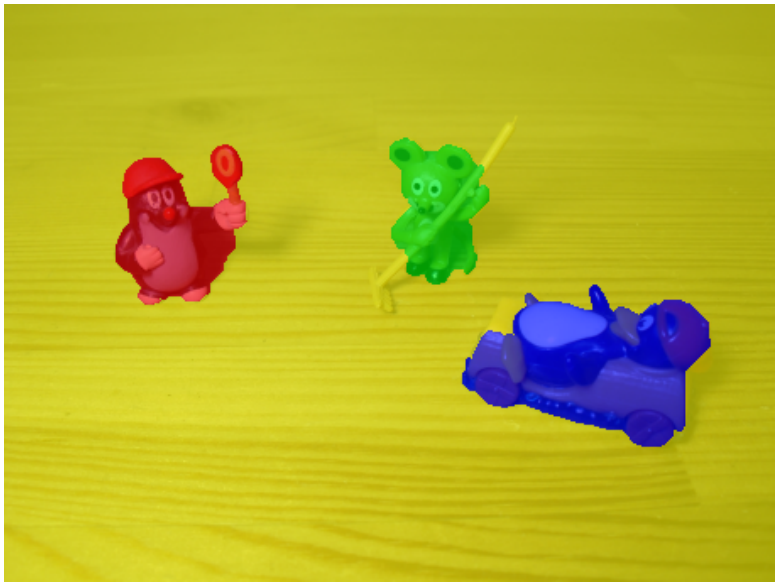


Figure 3.8. Results of the Δ-α – expansion algorithm applied on the output of the pure α – expansion shown in Figure 3.7 (b).

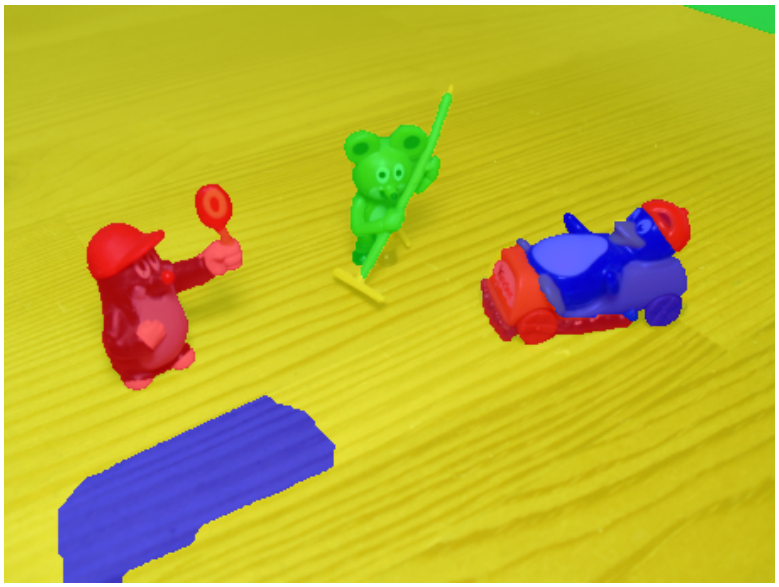


Figure 3.9. Results of Δ-α – expansion algorithm on the image that somehow differs from that where model was obtained. As it can be seen in this particular case Δ-α – expansion is not sufficient, we need some improvements.

4. Pixel swapping

As we described in Section 3, there can be cases when Δ-max-flow algorithm can fail and do not converge into desired interval. Of course solution might be just simple expand the interval, but what if after expanding the interval we relax the task in such a manner that we got bad results? As bad results we understand that there will be more than the expected number of targets or our target will not be segmented accurately enough (see Figure 1.1(c)).

For this particular case we developed algorithm that makes possible to achieve rather accurate changes, say at a pixel level, to gain the segmentation, we will call it *pixel swapping algorithm*.

Further in this work we will consider only border pixels as targets of potential changes. So we will deal with only two possible options:

1. algorithm is trying to expand current labeling f to fit its cardinality into requested interval

2. algorithm is trying to shrink current labeling f to fit its cardinality into requested interval

Pixel swapping algorithm

The whole algorithm 6 is described below:

Algorithm 6: Pixel swapping algorithm

Data: labeling f gotten from Δ-max-flow
unary and pairwise terms θ
Result: labeling f that satisfies object size constraint

```
/* H1 stores changes in energies after swapping foreground pixel to background    */
```

1. create priority queue H1;

```
/* H2 stores changes in energies after swapping background pixel to foreground    */
```

2. create priority queue H2;

```
/* H3 stores changes in energies after flipping edge between pixels with different
   labels                                                                         */
```

3. create priority queue H3;
4. δ = *find optimal change(H1, H2, H3)*;
5. apply found change δ;
6. modify priority queues;
7. if we have not met conditions on object size goto 4;
8. Return f;

Function *find optimal change* take the best choice over all priority queues and returns δ which is the parameter that tell us what changes should be applied to the current labeling f, *e.g.* swap pixel p from background to foreground or vice versa, flip an edge (change neighboring pixels labeling) e between two pixels p and q or swap pixel p from

foreground to background and simultaneously swap pixel q from background to foreground.

Description of the function *find optimal change* 7 is presented below:

Algorithm 7: Find optimal change

Data: priority queue H1
priority queue H2
priority queue H3

Result: parameter of changes δ

1. valueH1 = get root of H1;
2. valueH2 = get root of H2;
3. valueH3 = get root of H3;
4. valueH4 = bestH1 + bestH2; `// where bestH1 and bestH2 are the pair of pixels that are not neighbors in the image`
5. valuesArray = [valueH1, valueH2, valueH3, valueH4];
6. $i = \underset{i}{\text{argmax}}(indexof(max(valuesArray))), \quad i = 1, \ldots, 4$; `// i is the index of a maximum value in the values array`
7. set parameter δ according to i;
8. Return δ;

The main goal of this algorithm is to get target object into desired interval and at the same time try to keep energy as minimum as possible. It means that during run time of the algorithm the energy can slightly increase and our aim is to keep this change as small as possible.

Described algorithm uses three priority queues to store changes in energy after applying one of the following transformations, each for every queue:

1. swap foreground pixel to background
2. swap background pixel to foreground
3. flip edge between pixels with different labeling f

We compare elements inside queues using their values, that is an element with the highest change in energy (the lowest $|E_{change}|$ (4.1)) is placed in the root of the queue. Equation for computing an energy changes for one particular pixel p with labeling f_p is written in (4.1).

$$E_{temp} = E_{old} - \theta_{p_{f_p}} - \sum_{\{p_{f_p},q\}\in\mathbb{N}_{p_{f_p}}} \theta_{p_{f_p};q} + \theta_{p_{\bar{f}_p}} + \sum_{\{p_{\bar{f}_p},q\}\in\mathbb{N}_{p_{\bar{f}_p}}} \theta_{p_{\bar{f}_p};q}$$

$$E_{change} = E_{temp} - E_{old} \tag{4.1}$$

Here, $\mathbb{N}_{p_{f_p}}$ stands for the list of neighbors of the pixel p in the neighborhood grid (example of neighborhood grid is shown in Figure 2.1). $p_{\bar{f}_p}$ is the pixel p with the opposite label, *e.g.* for binary labels representation it holds, $\bar{f}_p = 1 - f_p$.

For each pixel in image we store energy changes in the corresponding priority queue, here as it can be seen from algorithm 6:

1. priority queue H1 stores changes when foreground pixel becomes background

2. priority queue H2 stores changes when background pixel becomes foreground

3. priority queue H3 stores changes when neighboring pixels with different labeling $f_p \neq f_q$ are flipped

The next step of the algorithm is finding an optimal change. After creating all priority queues we then proceed to finding the best swapping move. The algorithm goes iteratively the condition on the target object size is met. In each iteration we take root node from all the priority queues and additionally we create a new value that represents swapping non-neighbors pixels. The possible operations are written below:

1. take root r_1 from priority queue H1

2. take root r_2 from priority queue H2

3. take root r_3 from priority queue H3

4. compute energy change r_4 in case when we swap two non-neighbors pixels r_1 and r_2, where $f_{p_{r_1}} \neq f_{p_{r_2}}$ and $\{p_{r_1}, p_{r_2}\}$ does not exist

Here, p_{r_1} means pixel in image corresponding to energy change r_1.

We then compare obtained values and choose such a value that gives us the best change in energy, in other words a value that will minimize overall energy or increase it by as small as possible value. There can be four possible changes they are illustrated in Figure 4.1:

1. Figure 4.1 (a) shows how pixel with id 7 changes its labeling from foreground to background $f_{p_7} = 1 \rightarrow f_{p_7} = 0$

2. Figure 4.1 (b) shows how pixel with id 5 changes its labeling from foreground to background $f_{p_5} = 0 \rightarrow f_{p_5} = 1$

3. Figure 4.1 (c) shows edge flip between pixels with indexes 5 and 7 $f_{p_5} = 0 \rightarrow f_{p_5} = 1$ and $f_{p_7} = 1 \rightarrow f_{p_7} = 0$

4. Figure 4.1 (d) shows swapping labels between pixels with id 4 and 9 $f_{p_4} = 0 \rightarrow f_{p_4} = 1$ and $f_{p_9} = 1 \rightarrow f_{p_9} = 0$

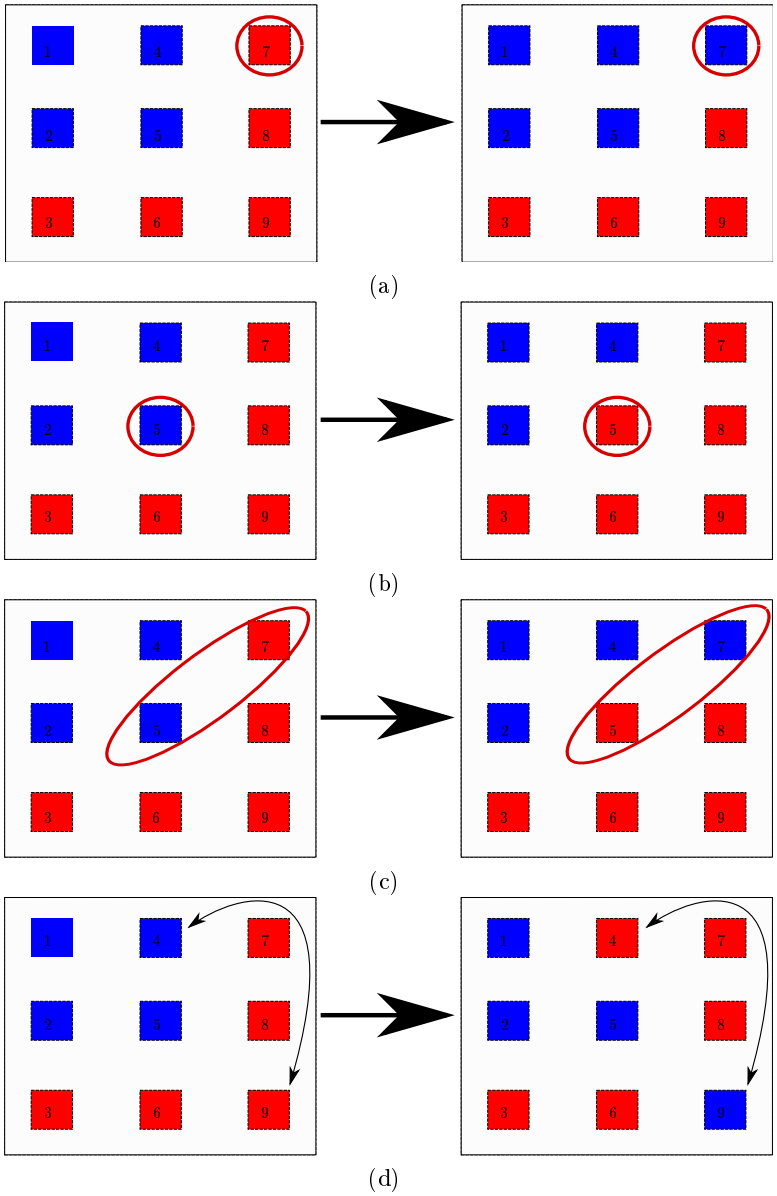


Figure 4.1. (a) shows the foreground to background swap. (b) shows the background to foreground swap. (c) shows the edge flip. (d) shows the swapping two non-neighbor pixels with different labeling. Note: red color corresponds to a foreground pixel, blue color corresponds to a background pixel.

In the next step we apply the best found change to our labeling f check if we are done, *i.e.* objects size constraint is met, if so then algorithm ends and output new labeling $\hat{f}$ in the other way it continues performing swapping.

In general algorithm is suitable for solving the following situations:

1. we have an initial labeling and we need to increase the number of foreground pixels 4.2 (b)
2. we have an initial labeling and we need to decrease the number of foreground pixels 4.2 (c)
3. we have an initial labeling and we need to eliminate some other false positive segmentations 4.2 (d)

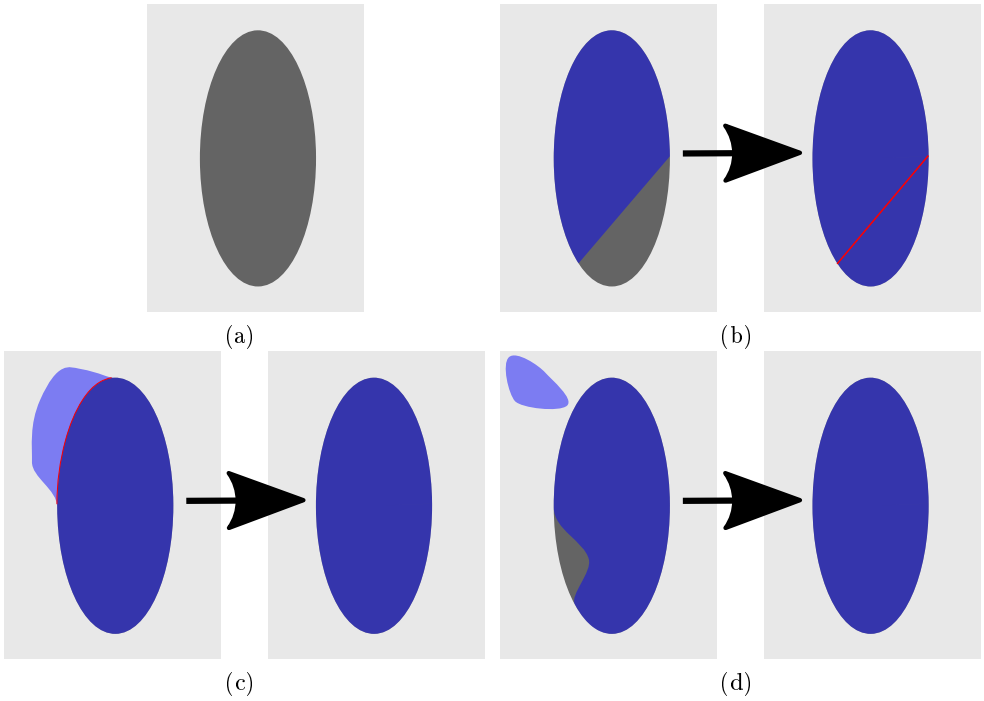


Figure 4.2. (a) shows object on which we want to apply segmentation algorithm. (b) algorithm is trying to expand found labeling. (c) algorithm is trying to decrease the number of foreground pixels. (d) algorithm is trying to eliminate the found false positive component (top left corner). Segmentation is defined by blue color.

In our case we are interested only in situations (a, b, c) illustrated in Figure 4.1, because we want to preserve only one component already found and just slightly modify it.

In Figure 4.4 is shown example of algorithm application. First of all we got our trained model from image illustrated in Figure 4.3. After obtaining the model we applied it to the image (a) (shown in Figure 4.4) and performed Δ-max-flow to acquire cardinality of labeling f within desired interval. Results are demonstrated in Figure 4.4 (b) it is clearly seen that the part of the object, the one in a shadow is not chosen by the algorithm. In the case that we are really needed these missing pixels be labeled as foreground we apply our swapping algorithm that will try to expand current cardinality of labeling f.

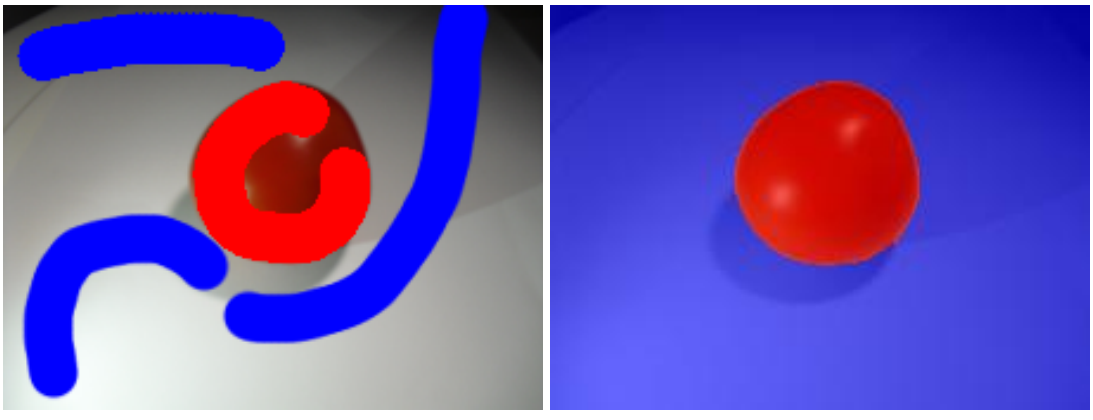
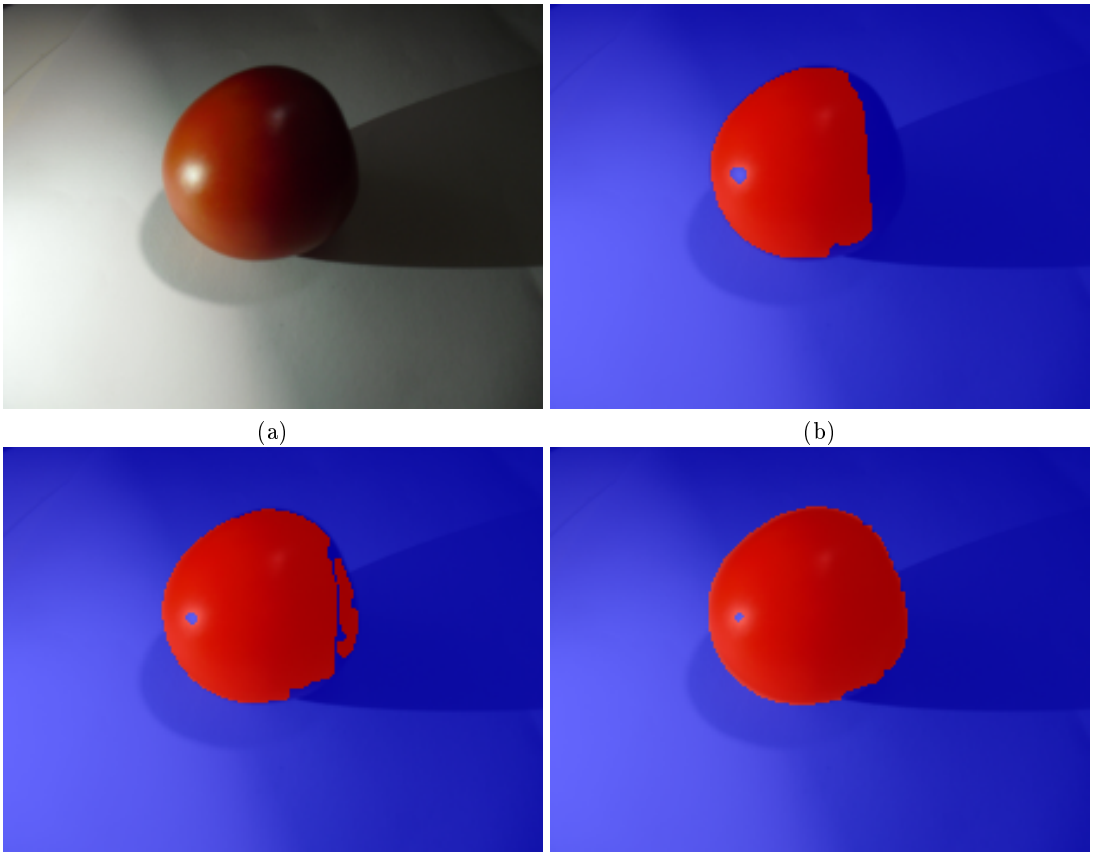


Figure 4.3. (a) shows the image from where color model was obtained. (b) shows segmentation based on pure max-flow.



(a) (b)

(c) (d)

Figure 4.4. (a) shows the image on which we applied trained model obtained from image 4.3 (a). (b) shows result of the Δ-max-flow algorithm. (c) shows a result after we run the swapping algorithm on the output of the Δ-max-flow. (d) is a result of the swapping algorithm and applied image dilatation on this result. Red color stands for the foreground and blue for the background.

One can see that in the image (c) in the Figure 4.4 some sort of a "scar" appears between the two foreground areas. It happened because the algorithm is needed to choose such a border pixel that will definitely increase total energy. Normally after performing segmentation we have the biggest difference between foreground and background pixels precisely on borders. So the algorithm tries to find such a value that causes minimal energy increasing. After this pixel was found the algorithm will expand to its neighbors and never return to those border pixels again. This behavior can be easily explain by fact that choosing neighbor pixels of a new foreground will lead to minimization of the overall energy, because these neighbor pixels will likely have smaller penalty (cost to change label from background to foreground) than initially found border pixels.

One can also eliminate such an output with "scars" simply using a dilatation on a result of swapping algorithm. Result of applying dilatation is shown in Figure 4.4 (d).

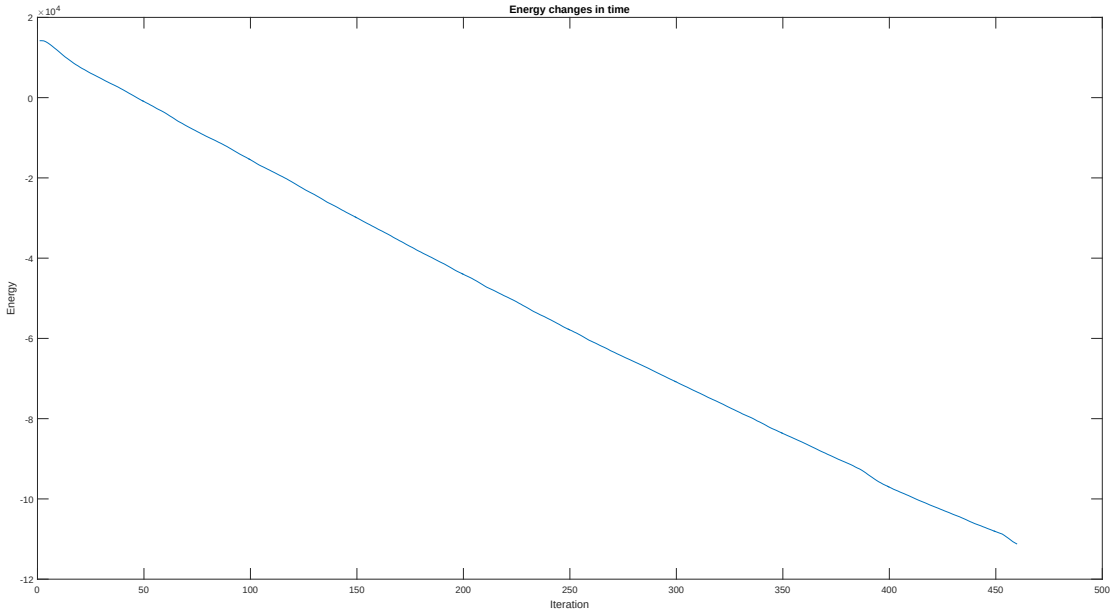


Figure 4.5. Energy changes during the swapping algorithm.

In Figure 4.5 is shown energy decreasing process. At the very beginning it can be seen that there is no change in energy. This is a moment where the algorithm chose new foreground pixel. After this the energy was decreasing nearly linearly.

5. Complexity

We now show that the binary energy minimization problem with object size constraint solving by the descried algorithms can also be solved by computing the graph bisection. Let us remind the energy function from Section 2 together with introduced global constraint on object size:

$$E(x) = \theta_{const} + \sum_{p \in \mathcal{V}} \theta_p(x_p) + \sum_{(p,q) \in \mathcal{E}} \theta_{pq}(x_p, x_q),$$
$$s.t. \quad \frac{\sum_{p \in \mathcal{P}} x_p}{|\mathcal{P}|} \in sz$$

here, $\sum_{p \in \mathcal{P}} x_p$ for binary problem gives us cardinality of foreground pixels set.

Graph bisection Let $G = (V, E)$ be an undirected graph with n vertices and m edges, where n is even. For a subset S of the vertices (with $S \neq \emptyset$ and $S \neq V$), the $cut(S, V \backslash S)$ is the set of all edges in G with one endpoint in S and one endpoint in $V \backslash S$; these edges are said to be the cut by $(S, V \backslash S)$. The *cost* of a *cut* is the number of edges in it.

A cut $(S, V \backslash S)$ is called a bisection of G if its two partitions, S and $V \backslash S$, are each of size $n/2$. Let us denote the minimum cost of a bisection of G by $b = b(G)$. *Minimum bisection* is the problem of computing b for an input graph G. This problem is known as NP-hard [5].

Based on knowledge that the graph bisection is NP-hard problem we prove that our binary energy minimization problem with object size constraint is NP-hard as well by reduction from *minimum bisection* to *the binary energy minimization problem with object size constraint*.

Reduction. Graph bisection $\rightarrow$ binary energy minimization problem with object size constraint We perform the following reduction to show that the binary energy minimization problem with object size constraint is NP-hard:

Let us assume that we have a graph $G = (V, E)$ that represents the graph bisection problem instance. We then transform it to a graph $G' = (V, E, \theta)$ that represents instance of the binary energy minimization problem with object size constraint. This transformation is shown in Figure 5.1.

In the next step we show that it is possible to transform instance of a graph G into a graph G' using the following operations:

1. for $\forall v \in V(G)$ we add unary terms vector θ_p, so vertex p now will have values for foreground and background labels
2. for $\forall e \in E(G)$ where e is an edge between two neighbor vertices $\{p, q\}$ we add pairwise terms vector θ_{pq} so edge e will now have penalty values for choosing different labels between $\{p, q\}$ and choosing same labels between $\{p, q\}$

Further we consider the fact that correct output of the graph bisection problem also can be described as the correct output of binary energy minimization problem with object

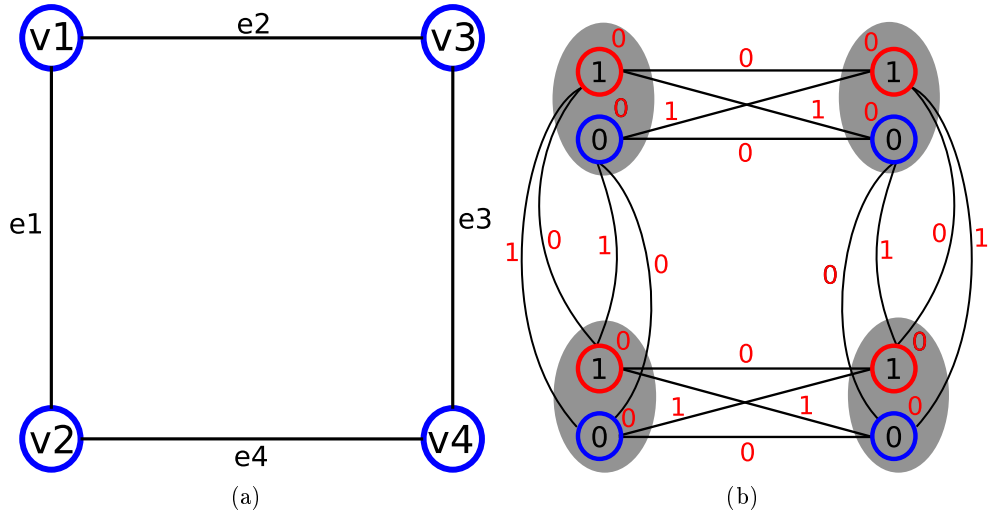


Figure 5.1. (a) shows an instance of the graph bisection problem (b) shows an instance of the binary energy minimization problem with object size constraint. Colors in graph (b) mean: red that pixel has foreground label and blue that pixel has background label.

size constraint. This is shown in Figure 5.2.

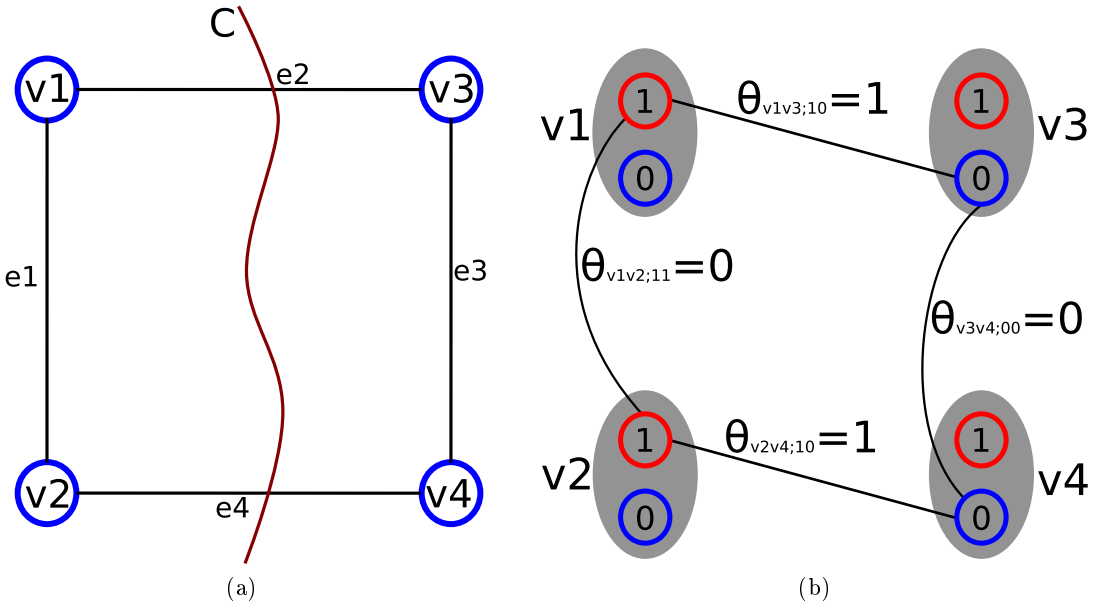


Figure 5.2. (a) shows correct output of the graph bisection. (b) shows correct output of the binary energy minimization problem with object size constraint. Colors in graph (b) mean: red that pixel has foreground label and blue that pixel has background label.

Approximation factor. We can also show that it is unlikely that the binary energy minimization problem with object size constraint has any constant factor approximation. Feige and Krauthgamer [4] showed that there exists a polylogarithmic approximation of the minimum bisection for planar graphs with ratio of $O(\log n)$ and for general graphs it is $O(log^{1.5}n)$ time of the known approximation ratio for min-ratio cuts.

6. Experimental results

In this section we will show results of applying designed algorithms and describe their weaknesses and strengths. We will show cases when these algorithms work perfectly and cases when additional effort is needed to improve quality. We will do an analysis of computational time and outline reasons why it is slow or fast.

6.1. Implementation details

All three algorithms:

1. Δ-max-flow
2. Δ-α – expansion
3. pixel swapping

were implemented in Matlab 2015b. For finding minimum cut in a graph we used the algorithm provided by Kolmogorov [10] also written in Matlab. For the *pixel swapping algorithm* were used priority queue. Unfortunately Matlab does not have build-in implementation of priority queue and it was somehow problematic to implement it due to Matlab specifics (vector oriented). So it was decided to use Java for priority queues implementation and connect compiled Java library with Matlab. The rest of the *pixel swapping algorithm* was written in Matlab.

6.2. System configuration

All algorithms were tested on the system with the following configuration:

Table 6.1. System configuration

CPU	Intel(R) Core(TM) i7-3517U CPU @ 1.90GHz
Memory	8.0 GB
OS	Windows 10 Home
Matlab	R2015b
JDK	1.7

6.3. Δ-max-flow

In this part we will show some results of applying Δ-max-flow algorithm.

As a first testing set were chosen batch of photos of Rotunda of St. Martin, at Vyšehrad in Prague. Photos were taken from different angles of view and approximately from the same distance. In these photos we chose as our target the door. We will call this set "rotunda".

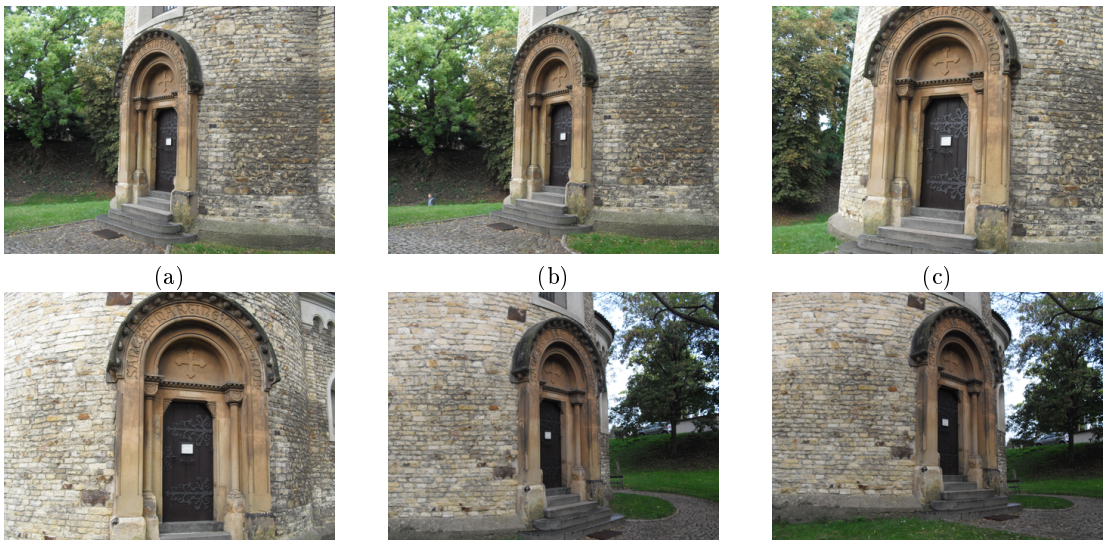
A next testing set consists of photographies of a mole toy with a simple background. We will show that it is enough to just train a model and apply it to another images from a set without using Δ-max-flow if the target object greatly differs from background and if the background is homogeneous (*e.g.* white). We will call this set "mole easy".

Third testing set will be again the mole toy but now we will put it to more complex background and also try to change light conditions. We will call this set "mole hard".

All the testing images had the following size:

$$width = 438px$$
$$height = 328px$$

Testing set "rotunda" The whole testing set is shown in Figure 6.1. As it was described in Section 3 we take first photo and train our color model on it. This photo and annotation is illustrated in Figure 6.2.



(a) (b) (c)

(d) (e) (f)

Figure 6.1. Testing set with photographies of the Rotunda of St. Martin, Vyšehrad, Prague.

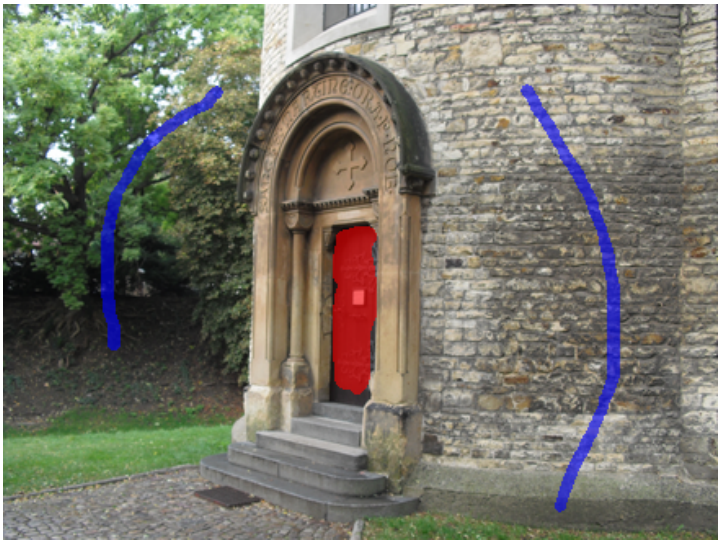


Figure 6.2. Image (a) from "rotunda" testing set with annotations.

As a first step we run *max-flow* algorithm on the first image from the set and get trained model with number of components $K = 1$. Then using the obtained model we perform segmentation of the rest of the set. Results are shown in Figure 6.3.

As it can be seen there are some errors for images (d) and (e), the algorithm

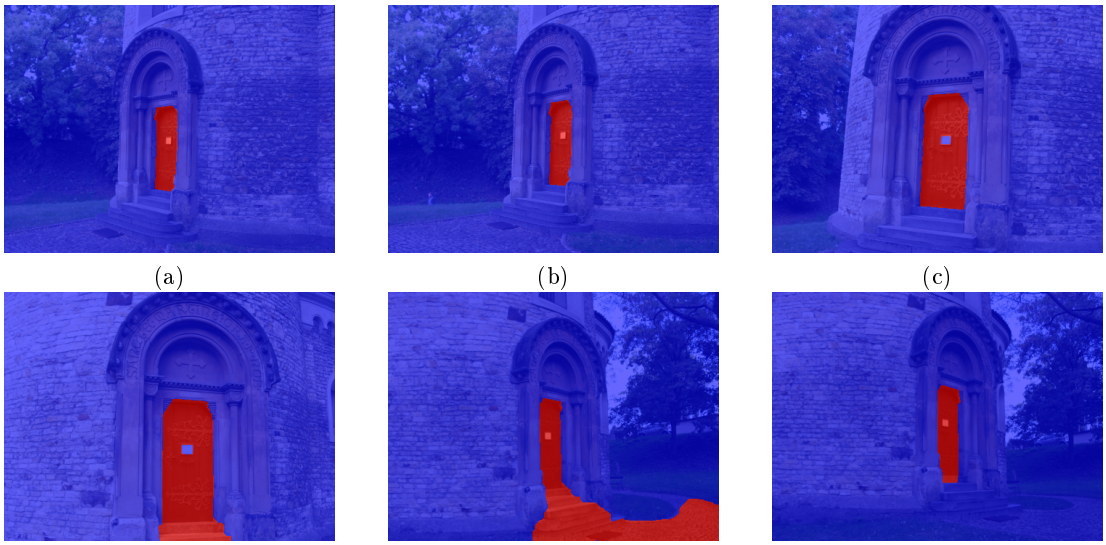


Figure 6.3. Segmentation results by using pure max-flow approach for the testing set "rotunda".

segmented a larger part than required. Such errors can be caused by different light conditions or by presence of an object that has similar colors.

We then run the Δ-max-flow algorithm on both images in order to acquire more accurate results. These results are shown in Figure 6.4.

Running time for generating model and performing segmentation using max-flow and Δ-max-flow algorithms for the "rotunda" testing set are shown in Table 6.2.

(a) (b)

Figure 6.4. Results of the Δ-max-flow applied to images (d) and (e) from the "rotunda" testing set. Algorithm fits cardinality of f into the interval [2 5]. Image (a) corresponds to the image (d) from Figure 6.3 and (b) corresponds to the image (e) from Figure 6.3.

Table 6.2. Running time for the testing set "rotunda". Δ-max-flow

Image	Training a model[s]	Max-flow segmentation[s]	Δ-max-flow[s]
(a)	9.436	1.566	-
(b)	-	1.478	-
(c)	-	1.475	-
(d)	-	1.465	5.50912
(e)	-	1.577	6.36154
(f)	-	1.794	-

In the case of "rotunda" testing set Δ-max-flow algorithm successfully fulfill all requests with a high quality output.

Testing set "mole easy" The whole testing set is shown in Figure 6.5. The image with an annotation on which we perform training is demonstrated in Figure 6.6. Model was trained with a number of components $K = 1$.

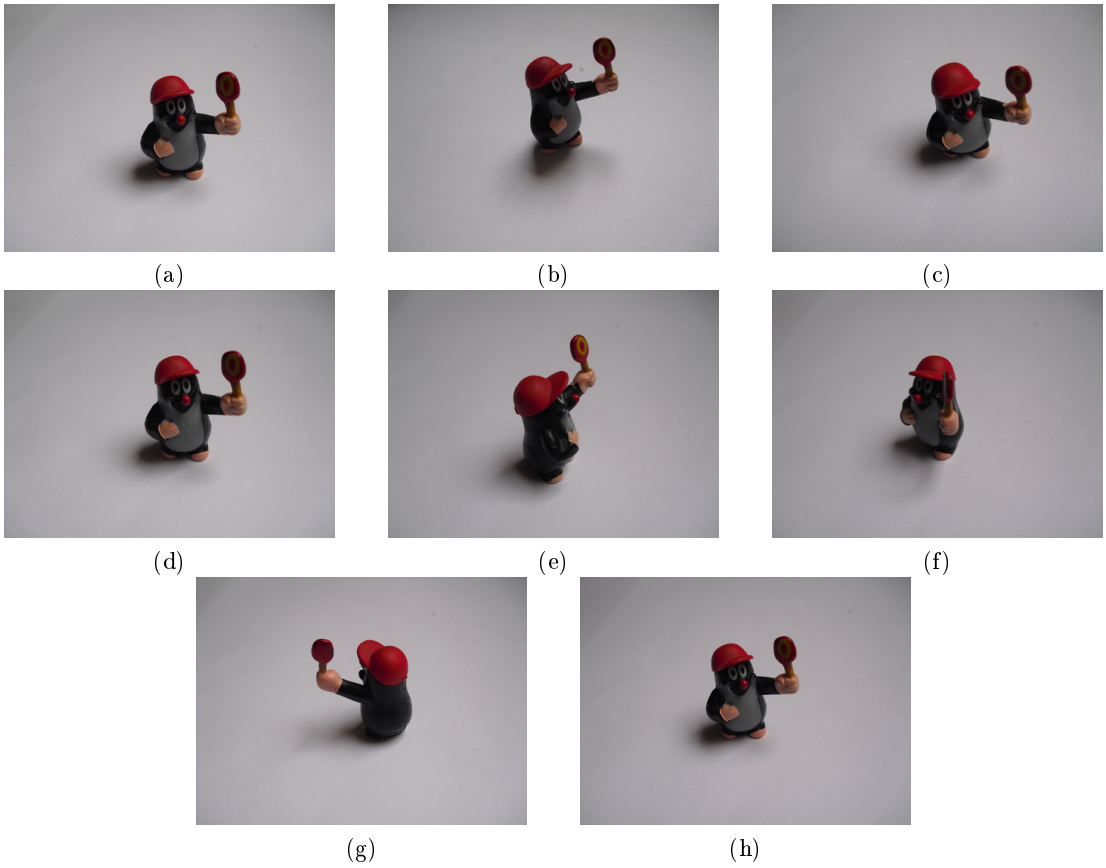


Figure 6.5. Testing set with photographies of the mole toy with homogeneous background.

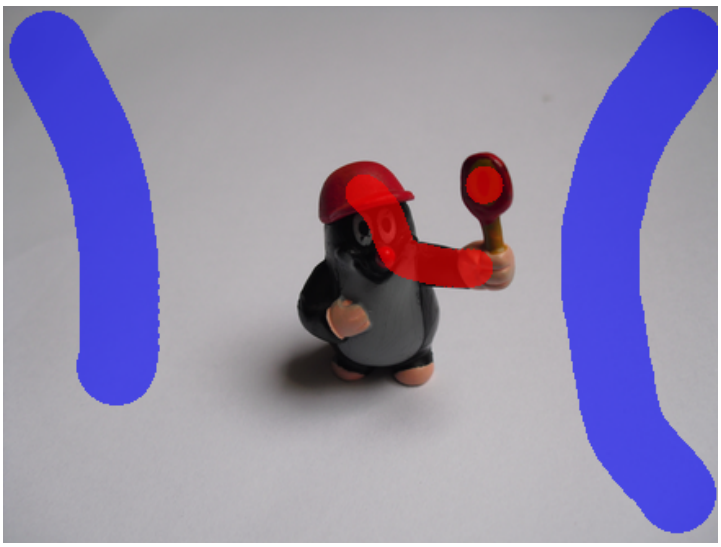


Figure 6.6. Image (a) from "mole easy" testing set with annotations.

There is one interesting thing for this particular testing set. After obtaining a model and performing classic max-flow segmentation we have already gotten correct results, that is we did not need to perform any further optimizations. These results are shown in Figure 6.7.

It means that theoretically if we have a homogeneous background as in the example

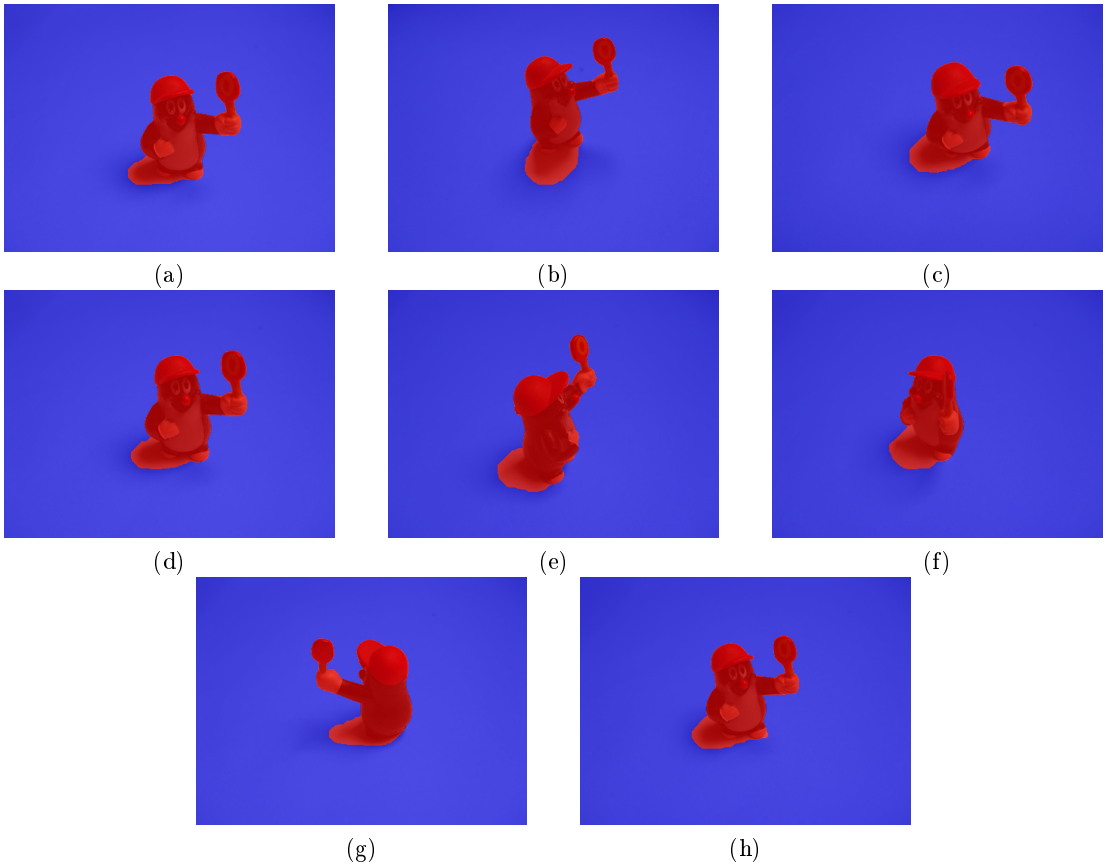


Figure 6.7. Segmentation results by using pure max-flow approach for the testing set "mole easy".

above and target object that greatly differs from background one can simple use acquired model on the whole set.

Running times for generating model and performing segmentation using max-flow are shown in Table 6.3. We do not perform Δ-max-flow on any of the image from this training set because it was not necessary.

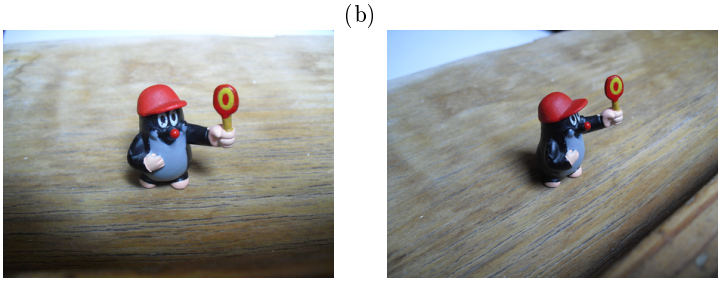
Testing set "mole hard" The whole testing set is shown in Figure 6.8. The image with an annotation on which we perform training is demonstrated in Figure 6.9. Model was trained with a number of components $K = 30$. We use such a huge number of components here in order to obtain better model that can be capable of distinguishing between our target and background.

It can be seen that image (d) and (e) differ from the rest of the set. It will be interesting to see how can our approach deal with a such situation. First of all we perform classical segmentation using gained model and max-flow approach. These results are illustrated in Figure 6.10.

Table 6.3. Running time for the testing set "mole easy". Δ-max-flow

Image	Training a model[s]	Max-flow segmentation[s]	Δ-max-flow[s]
(a)	10.217	1.166	-
(b)	-	1.460	-
(c)	-	1.437	-
(d)	-	1.453	-
(e)	-	1.447	-
(f)	-	1.437	-
(g)	-	1.412	-
(h)	-	1.480	-

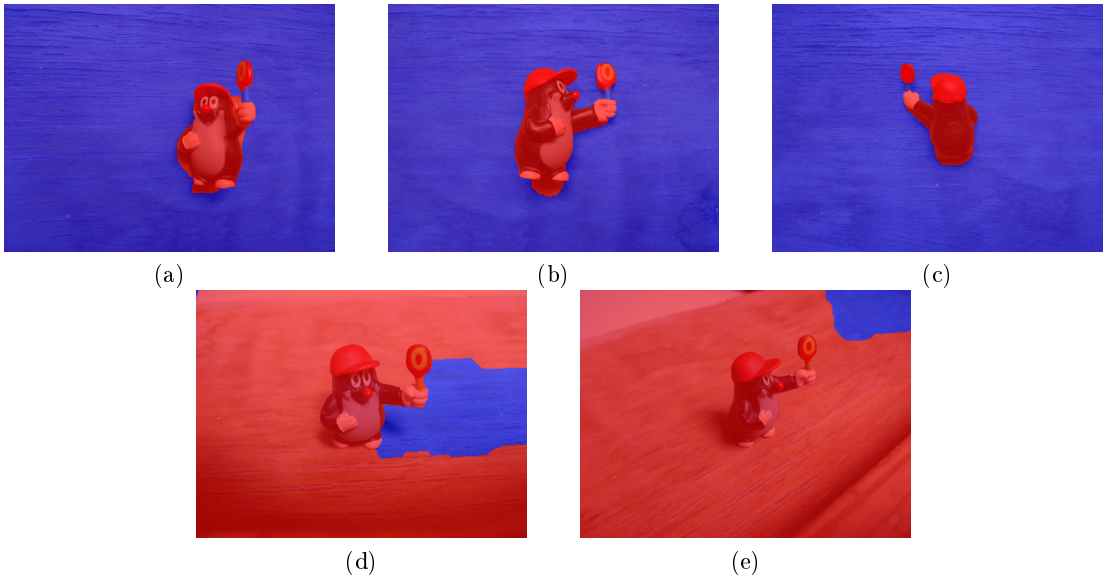
(a) (b) (c)



(d) (e)

Figure 6.8. Testing set with photographies of the mole toy with quite complex background and with different light conditions.

Figure 6.9. Image (a) from "mole hard" testing set with annotations.



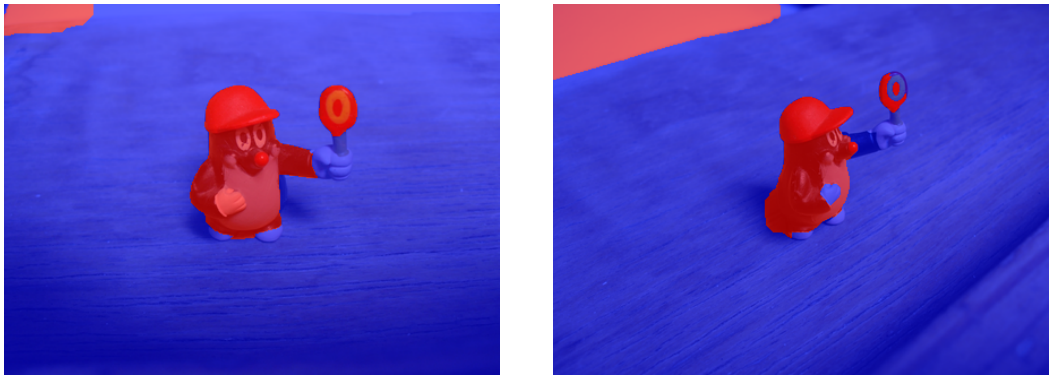
(a) (b) (c) (d) (e)

Figure 6.10. Results of classical max-flow segmentation using trained model.

As it could be expected classical max-flow approach failed with images (d) and (e) from Figure 6.8. Different light, target position and "new background" significantly affected the scene and classical max-flow was hardly 'confused".

Now we apply Δ-max-flow on Figure 6.8 (d) and Figure 6.8 (e). Results are shown in Figure 6.11.

Surprisingly it performed very well for such a simple approach. While here are still



(a) (b)

Figure 6.11. Results of the Δ-max-flow applied to images (d) and (e) from the "mole hard" testing set. Algorithm fits cardinality of f into the interval [9 11]. Image (a) corresponds to the image (d) from Figure 6.10 and (b) corresponds to the image (e) from Figure 6.10.

some incorrectly labeled parts it is already possible to use the gained output. It also make further improvements possible, *e.g.* adding new constraint for numbers of searched target.

Unfortunately to get sensible output we need to pay cost in computational time. Running time for generating model and performing segmentation using max-flow and Δ-max-flow algorithms for the "mole hard" testing set are shown in Table 6.4.

Table 6.4. Running time for the testing set "mole hard". Δ-max-flow

Image	Training a model[s]	Max-flow segmentation[s]	Δ-max-flow[s]
(a)	340.705	2.966	-
(b)	-	2.263	-
(c)	-	2.298	-
(d)	-	3.268	8.282
(e)	-	3.781	8.323

In the case of "mole hard" testing set Δ-max-flow algorithm performed notably well. It demonstrated ability to handle different light conditions and handle presence of previously unknown background.

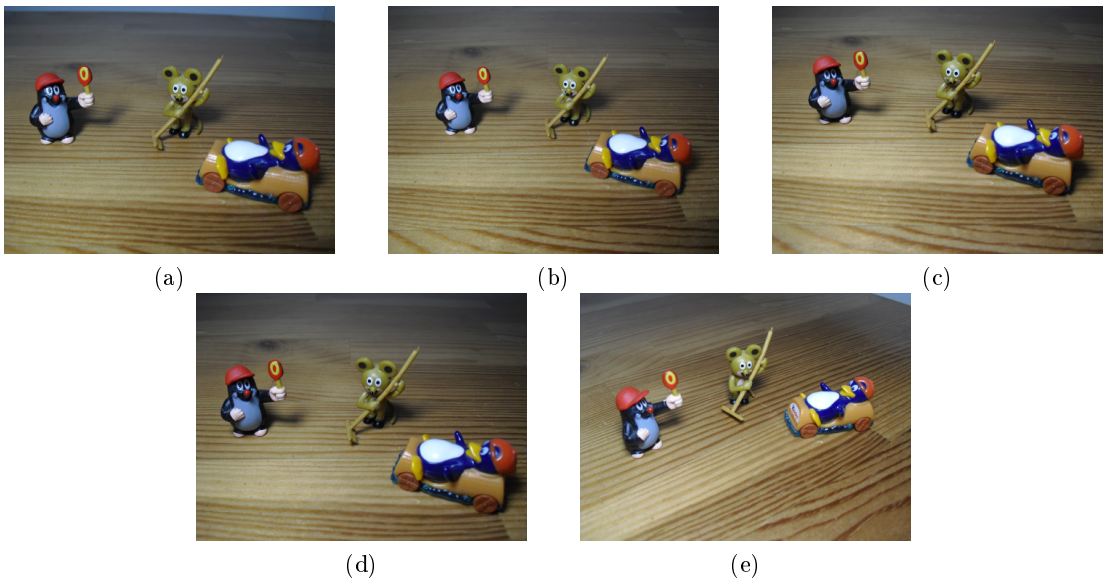
Conclusion of the Δ-max-flow Δ-max-flow showed its usefulness in different situations with relatively fast computational time.

6.4. Δ-α – expansion

In this part we will show some results of applying Δ-α – expansion algorithm to the image set. As a testing set were chosen batch of photos of different toys. Photos again were taken from different angles of view and approximately from the same distance to preserve objects size. We are interested in segmentation of three different toys: mole, mouse and penguin. The whole testing set are shown in Figure 6.12. We will call this set "toys".

All the testing images had the following size:

$$width = 456px$$
$$height = 342px$$



(a) (b) (c) (d) (e)

Figure 6.12. Testing set with photographies of different toys with non-homogeneous background.

Model was obtained from the first image from the set. This image with annotation is shown in Figure 6.13. Our model was trained with a number of components $K = 35$. While dealing with multiple labels it is always necessary to acquire "well trained" model, here with large K, to be capable of distinguishing between many targets.

Next we will show how classical α – expansion algorithm performed with the rest of

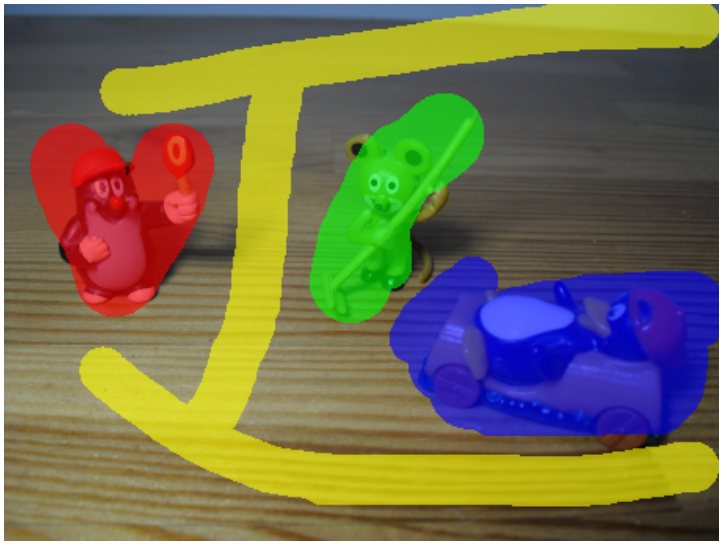


Figure 6.13. Image (a) from "toys" testing set with annotations.

the set using obtained model. Results are shown in Figure 6.14.

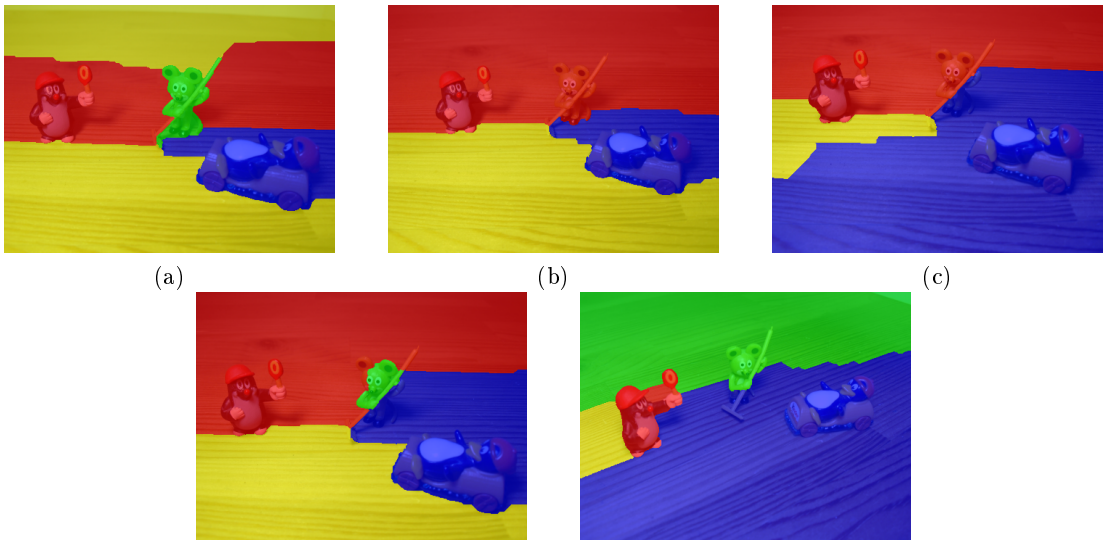
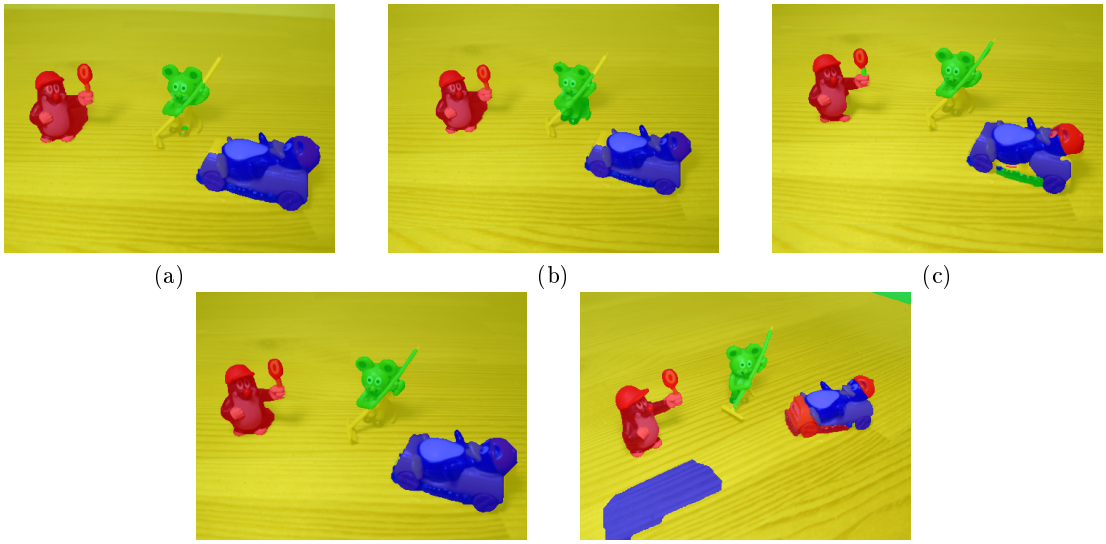


Figure 6.14. Results of applying α – expansion algorithm to the "toys" testing set.

It is not hard to see that pure α – expansion can not handle such a task. Output is very raw, inaccurate segmentation, even for the image (a) from which we have gained the model. The alpha expansion algorithm does not lead to globally optimal solution. Hence, ambiguities can appear in the results, especially when we apply model gained from different image.

Next we will try to apply our Δ-α – expansion approach on this testing set. Results are shown in Figure 6.15.

In compare to results from pure α – expansion we got pretty accurate output with



(a) (b) (c) (d) (e)

Figure 6.15. Results of applying Δ-α – expansion algorithm to the "toys" testing set.

the exception of few small details. It can be seen that in image (e) in Figure 6.15 did not manage to correct annotate penguin toy. It can be caused by insufficient information in model about color of the "car" on which this penguin was placed. It can be also spotted small green annotation in the right top corner. This can happen when there is no information in the model about a new background.

Regardless of some errors one can see that the Δ-α – expansion did a nice job segmenting each individual toy.

Table 6.5 shows running times for obtaining model and performing α – expansion and Δ-α – expansion.

Table 6.5. Running time for the testing set "toys". Δ-α – expansion algorithm.

Image	Training a model[s]	α – expansion segmentation[s]	Δ-α – expansion[s]
(a)	223.713	12.406	220.939
(b)	-	16.293	210.035
(c)	-	12.298	313.397
(d)	-	11.228	202.149
(e)	-	13.181	241.058

Conclusion of the Δ-α – expansion Δ-α – expansion showed that is possible to enhance pure output of classical α – expansion and achieve better results but for the cost of increasing computational time.

6.5. Pixel swapping

In this section we will show example of applying *pixel swapping algorithm* for obtaining more accurate results.

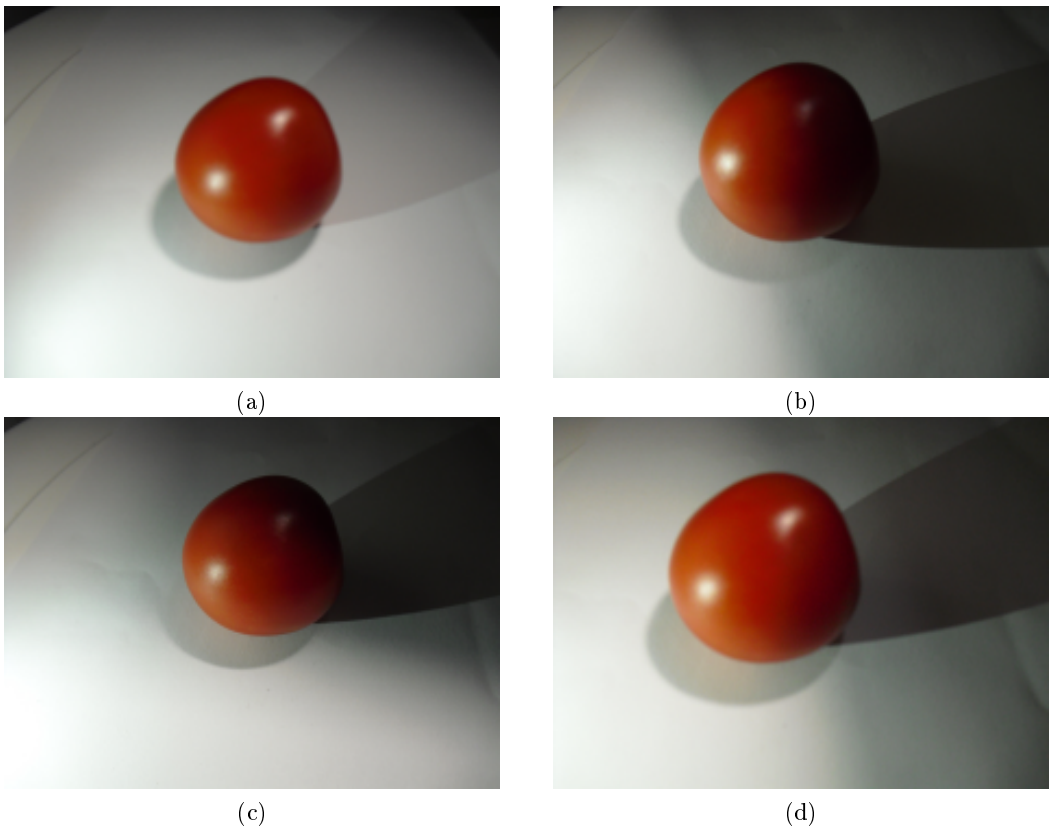
As a testing set was chosen photographies of tomato lying on white paper. For this experiment we took one photo with just tomato and to the rest of the set we added additional shadows to make the segmentation task harder for our Δ-max-flow algorithm.

The whole set is shown in Figure 6.16. We will call this "tomato".

All the testing images had the following size:

$$width = 228px$$
$$height = 171px$$

Model was obtained from the first image from the set. This image with annotation



(a) (b) (c) (d)

Figure 6.16. Testing set with photographies of a tomato with different shadow conditions.

is shown in Figure 6.17. This model was trained with a number of components $K = 1$.

As for the previous cases we first perform classical segmentation using max-flow with trained model and see if there are some improvements needed to be applied to results. The output of the segmentation using max-flow approach is shown in Figure 6.18.

As it can be seen images (b), (c), (d) from Figure 6.16 were not perfectly segmented

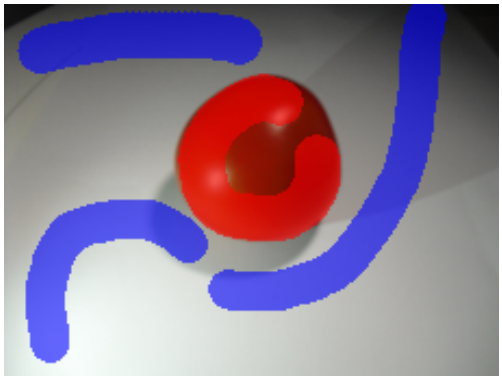
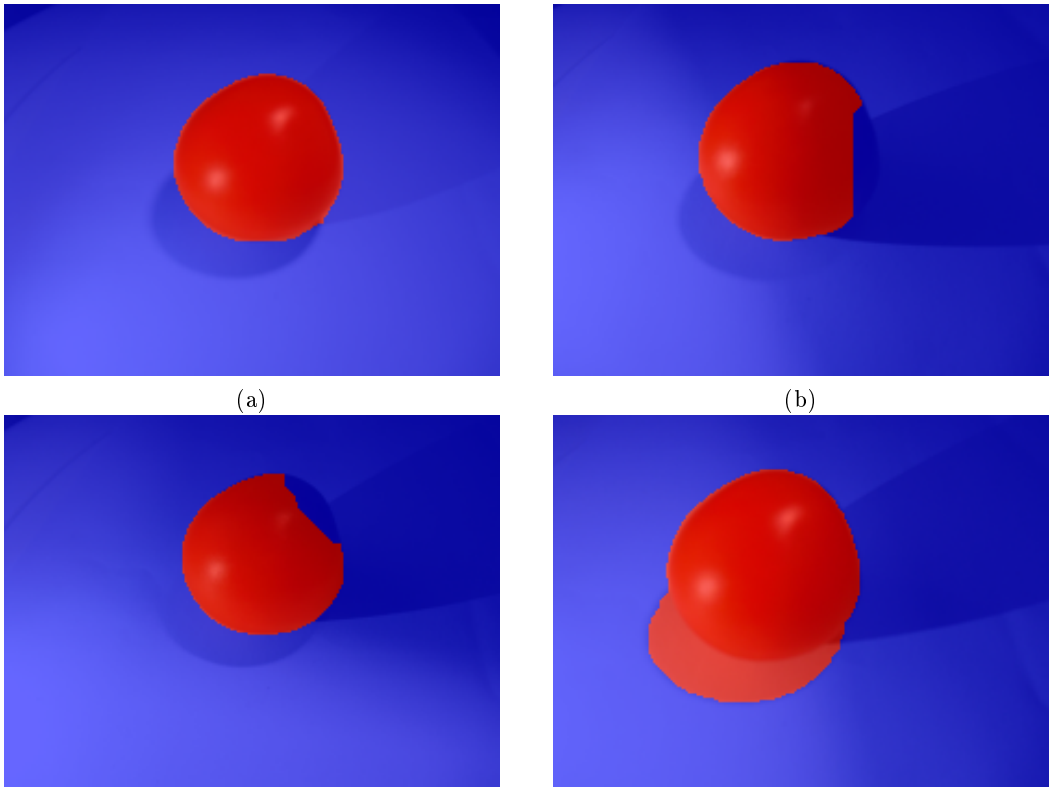


Figure 6.17. Image (a) from "tomato" testing set with annotations.

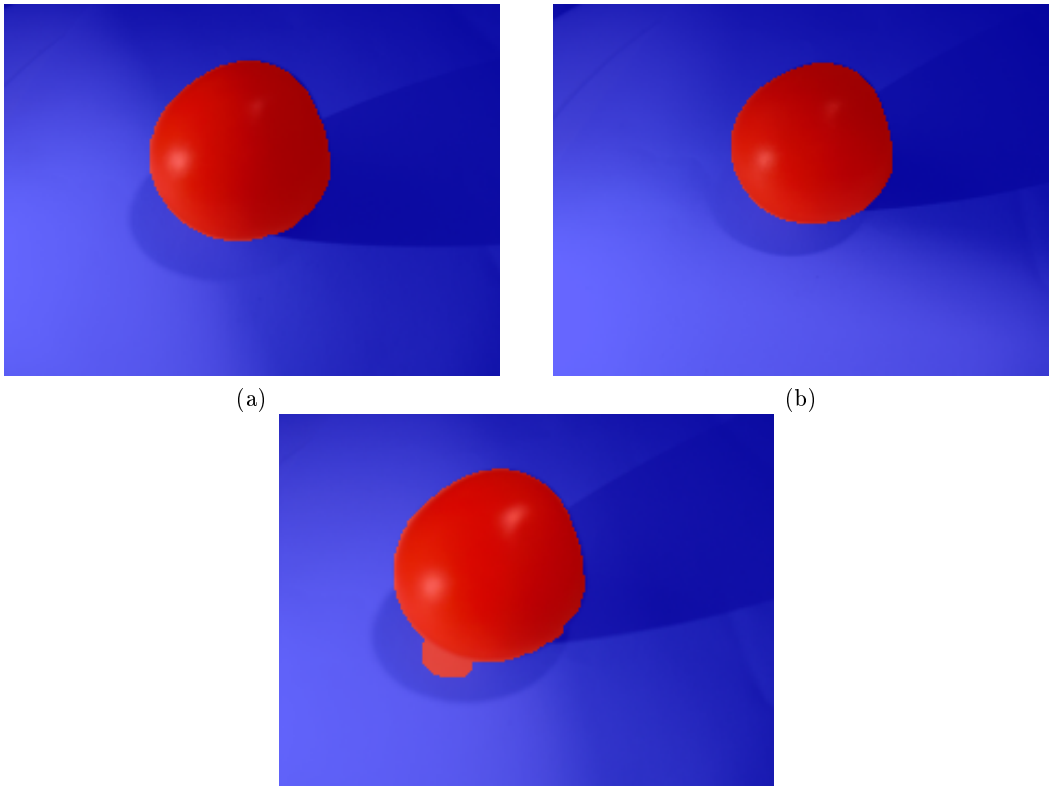


(a) (b)

(c) (d)

Figure 6.18. Results of doing segmentation using classical max-flow method.

by using just trained model. It mainly cause by shadows and inability of classical max-flow handle this situation. So we run our Δ-max-flow algorithm on these images to see if there would be better output. Results are shown in Figure 6.19.



(a) (b)

(c)

Figure 6.19. Results of Δ-max-flow applied to images (b), (c), (d) from Figure 6.18. The image (a) corresponds to the image Figure 6.18(b), the image (b) corresponds to the image Figure 6.18(c) and the image (c) corresponds to the image Figure 6.18(d).

We see that results now are much better but still there is small misclassification in the image (d) of Figure 6.19. Part of the shadow was also classified as our target object. When these cases happen in order to obtain more authentic output one can use *pixel swapping* algorithm. In Figure 6.20 is illustrated the result of applied *pixel swapping* algorithm to the image Figure 6.19 (d).

We can see that output in now pretty accurate and fulfills our requests.

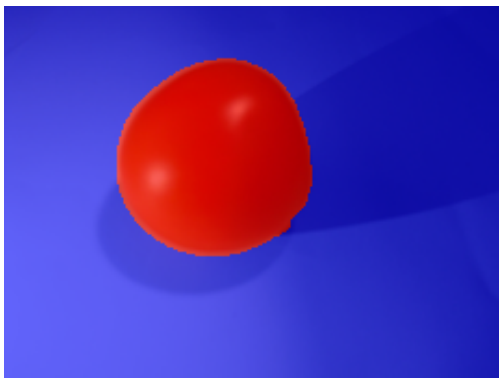


Figure 6.20. The result of applying *pixel swapping* algorithm on image (c) from Figure 6.19.

Table 6.6. Running time for the testing set "tomato". *Pixel swapping* algorithm

Image	Training a model[s]	Max-flow[s]	Δ-max-flow[s]	Pixel swapping
(a)	2.15471	12.406	-	-
(b)	-	0.420	3.706	-
(c)	-	0.389	3.221	-
(d)	-	0.407	2.504	89.748

In Table 6.6 are shown running times for classical max-flow, Δ-max-flow and *pixel swapping* algorithm.

Conclusion of the pixel swapping We demonstrated that it is possible to improve accuracy of the segmentation by applying *pixel swapping* algorithm but with growing computational cost.

7. Conclusion

In this thesis we have described state of the art methods applied for energy minimization task. Image segmentation task was chosen as a problem of energy minimization. The goal of this task is to split pixels in image that have similar color into several sets by using user provided initial annotation.

We used obtained GMM model from the one of the image with known annotations and apply it to the similar images within the same dataset.

It was shown that is just not enough to simple apply obtained GMM model to the rest of the set without losing accuracy of the segmentation.

We introduced global constraints as relaxations to problem of the energy minimization. As follows three fast (approximation) algorithms that uses global constraints were also introduced. These are the following algorithms:

1. Δ-max-flow
2. Δ-α – expansion
3. pixel swapping

Δ algorithms (Δ-max-flow and Δ-α – expansion) use special auxiliary parameter Δ to adjust unary weights of each labels and fit cardinality of a target object into the desired interval. The best Δ is found by using bisection interval method as optimization approach.

Pixel swapping is an iterative algorithm based on knowledge of changes in energies after swapping some of pixel was performed. Main goal as in cases of Δ algorithm is fitting searched object into desired interval. It uses priority queues to store energy changes and always tries to apply the best one change therefore minimizing overall energy. This algorithm is applicable only after Δ algorithm fails.

Algorithms performance was evaluated on five different datasets (rotunda of St. Martin, at Vyšehrad, mole toy on a homogeneous background, mole toy on a complex background, bunch of toys on a complex background and tomato on a homogeneous background) and compared with classical approach for the segmentation task (max-flow and α – expansion) with just simple applying acquired model.

It was shown that computational cost is growing with applying global constraints to the task, especially for Δ-α – expansion algorithm.

We showed that the binary energy minimization problem with object size constraint is NP-hard by reduction the graph bisection problem, known to be NP-hard [5] to our problem. Besides we can not expect existence of constant approximation factor.

7.1. Future Work

A limitation of the current approach is that it can not handle situations when multiple target components (disjoint areas) with the same labels appear and high computational time can be required due to some Matlab limitations.
Possible extensions can be to introduce new global constraints that make situation with several target components possible to solve. This can be *e.g.* number of components.

Perhaps methods that can compute homogeneity of components may be useful for this. It also possible to improve computational time by re-written these algorithm into C/C++. We can also improve pixel swapping algorithm by expanding it to more labels in the same way as it was for binary problem. We can iterate over all possible labels and try to adjust them to required interval but it will probably need better data structures.

A. Contents of the enclosed DVD

- /
  - thesis
    - thesis.pdf ..... digital copy of this thesis
  - application ..... implementation of described algorithms
    - main_rotunda.m ..... will run Δ-max-flow on “routunda“ dataset
    - main_mole_easy.m ..... will run Δ-max-flow on “mole easy“ dataset
    - main_mole_hard.m ..... will run Δ-max-flow on “mole hard“ dataset
    - main_toys.m ..... will run Δ-α – expansion on “toys“ dataset
    - main_tomato.m ..... will run Δ-max-flow and pixel swapping on “tomato“ dataset
  - data ..... testing data thesis
    - rotunda ..... “routunda“ dataset
    - mole_easy ..... “mole easy“ dataset
    - mole_hard ..... “mole hard“ dataset
    - toys ..... “toys“ dataset
    - tomato ..... “tomato“ dataset

B. User manual

Installation

There is no installation needed. However it is necessarily that Matlab 2015b or higher and Java JDK 1.7 or higher must be installed. To run provided algorithms one should simple copy folders "application" and "data" from DVD to a local folder and run appropriate scripts described in Appendix A.

Algorithms output

All output results of algorithms would be saved as images to "data" folder. For each testing dataset results will be stored in a corresponding folder, *e.g.* for the dataset "rotunda" it will be:

1. "data/rotunda/results_pure" for classical max-flow
2. "data/rotunda/results_delta" for Δ-max-flow
3. "data/rotunda/1_initial.png" shows annotated image which was used for obtaining model

Bibliography

[1] Kolmogorov V., Rother C. Minimizing nonsubmodular functions with graph cuts - a review. *IEEE Trans. on Pattern Recognition and Machine Intelligence*, 29(7), JULY 2007. ix, 1, 7, 8

[2] Šonka M., Hlaváč V., Boyle R. *Image Processing, Analysis and Machine vision.* Thomson Learning, Toronto, Canada, 3rd edition, 2007. 1, 5

[3] Boykov Y., Veksler O., Zabih R. Fast approximate energy minimization via graph cuts. *IEEE Trans. on Pattern Recognition and Machine Intelligence*, 23, November 2001. ix, 6, 9, 10

[4] Feige U., Krauthgamer R. A polylogarithmic approximation of the minimum bisection. *Society for Industrial and Applied Mathematics*, 31(4):1090–1118, 2002. 27

[5] Garey M. R., Johnson D. S., and Stockmeyer L. Some simplified NP-complete graph problems. *STOC '74 Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 47–63, MAY 1974. 26, 43

[6] Gupta R., Diwan Ajit A., Sarawagi S. Efficient inference with cardinality-based clique potentials. *Appearing in Proceedings of the 24th International Conference on Machine Learning*, 2007. 3

[7] Kernighan B. W., Lin S. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, February 1970. 4

[8] Tarlow D., Givoni Inmar E., Zemel Richard S. Hop-map: Efficient message passing with high order potentials. *Appearing in Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, 9, 2010. 3

[9] Werner T. Revisiting the linear programming relaxation approach to Gibbs energy minimization and weighted constraint satisfaction. *IEEE Trans. on Pattern Recognition and Machine Intelligence*, 2010. 3

[10] Kolmogorov V. Max-flow algorithm implementation. `http://pub.ist.ac.at/~vnk/software.html` [Online]. 1, 28

[11] Delong A., Gorelick L., Veksler O., Boykov Y. Minimizing energies with hierarchical costs. In *International Journal of Computer Vision*, volume 100, pages 38–58. Springer, 2012. 3