Department of Mathematics and Computer Science
Distributed Systems Group
Prof. Dr. B. Freisleben

SEMI-AUTOMATIC, GRAPH-BASED VERTEBRA SEGMENTATION IN MRI DATA

by

Robert Schwarzenberg

Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of
BACHELOR OF SCIENCE
in
COMPUTER SCIENCE

Title:

Semi-Automatic, Graph-Based Vertebra Segmentation in MRI Data

Author:

Robert Schwarzenberg
Hofstatt 19/20
35037 Marburg
Germany

Submission:

October 5, 2012
University of Marburg, Marburg, Germany
Department of Mathematics and Computer Science

Examining supervisor:

Prof. Dr. Bernd Freisleben
University of Marburg, Marburg, Germany
Department of Mathematics and Computer Science

External supervisor:

Dr. Dr. Jan Egger
Harvard Medical School, MA, USA
Brigham and Women's Hospital
Department of Radiology

.

University of Marburg, Marburg, Germany
Department of Mathematics and Computer Science
Department of Medicine

Abstract

The subject area of this thesis primarily covers vertebral body segmentation in volumetric MRI data, however the algorithm presented here also targets other cubic-shaped, anatomical structures. Automatic, computer-aided vertebra segmentation aims to decrease the time physicians spend on the preoperative evaluation of surgical patients.

The foremost problems in the field of vertebral segmentation are weak boundaries and outliers inside the structure. The newly developed approach tackles these difficulties by using a cubic-shaped template: it allows the user to impose a smoothness constraint Δ on the segmentation result, where a Δ-value of zero results in a regular, cubic shape, while a Δ-value greater than zero permits a corresponding deviation.

For this, the algorithm generates a weighted, two-terminal graph $G = (V(G), E(G))$ (s-t-network) within the cubic template. After its construction, the minimal closed set on the graph is computed via a polynomial time s-t-cut, creating a 3D segmentation of the vertebral body.

The nodes $v \in V(G)\backslash\{s,t\}$ correspond to voxels that are distributed along a number of rays which extend from a user-defined seedpoint inside the vertebral body and intersect with its outer boundaries.

Each vertex is connected to a virtual source s and a virtual sink t and the capacities of the corresponding edges reflect a node's affiliation with the source (vertebra) and the sink (background). Furthermore, the algorithm sets up a set of infinity-weighted edges, connecting the vertices on a ray and a set of infinity-weighted inter-ray edges. The latter implements the smoothness constraint while the first set ensures that each ray is cut exactly one time, ideally right in front of the outer boundaries.

An evaluation of the algorithm led to an average DSC of 81.88%, ranging from 71.64% to 86.69%. The computationally most intensive parameter settings resulted in a processing time of 19.1 seconds (2.1 GHz, 4 GB RAM).

Zusammenfassung

In dieser Arbeit wird ein neuer Ansatz zur dreidimensionalen Wirbelkörpersegmentierung in MRT Aufnahmen präsentiert, der sich auch zur Segmentierung anderer, würfelförmiger anatomischer Strukturen eignet. Die automatische Segmentierung von Wirbeln zielt darauf ab, die Zeit, die Mediziner für die präoperative Evaluierung von Patienten aufwenden, zu verkürzen.

Im Kontext von Wirbelsegmentierungen stellen homogene Objekt-Hintergrundübergänge sowie Ausreißer innerhalb der anatomischen Struktur die größten Herausforderungen dar. Der hier präsentierte Algorithmus begegnet dem, indem er das Segmentierungsergebnis einer würfelförmigen Vorgabe annähert: Dem Nutzer ist es möglich, den Grad Δ der Abweichung des Segmentierungsergebnisses von einer Würfelform zu bestimmen. Ist Δ gleich null, so nimmt die Segmentierung eine reguläre Würfelform an, bei einem Δ-Wert über null sind entsprechende Abweichungen möglich.

Hierzu generiert der Algorithmus einen gerichteten zweiterminalen Graph $G = (V(G), E(G))$ (s-t-Netzwerk) innerhalb der würfelförmigen Vorgabe. Nach der Konstruktion wird in polynomialer Laufzeit ein minimaler s-t-Schnitt berechnet, der die Knoten $v \in V(G)$ in zwei disjunkte Mengen teilt, aus denen dann die Segmentierung ermittelt wird. Die Knoten $v \in V(G)\backslash\{s, t\}$ entsprechen Voxeln, die sich entlang mehrerer Strahlen in der MRT-Aufnahme verteilen, wobei alle Strahlen ihren Ursprung in einem benutzerdefinierten Saatpunkt haben und die Außengrenzen des Wirbelkörpers durchstoßen.

Jeder Knoten ist zudem über jeweils eine Kante $e \in E(G)$ mit einer virtuellen Quelle s und einer virtuellen Senke t verbunden. Die Kapazitäten der entsprechenden Kanten reflektieren hierbei die Affinität eines Knoten zur Quelle (Wirbel) und zur Senke (Hintergrund). Außerdem konstruiert der Algorithmus eine Menge von unendlich gewichteten Kanten, die die Knoten auf einem Strahl untereinander verbinden und eine Menge unendlich gewichteter Kanten, die Knoten, auf verschiedenen Strahlen verbinden. Die letztere Menge implementiert die benutzerdefinierte Abweichung, während die erstbeschriebene sicherstellt, dass jeder Strahl genau einmal geschnitten wird, idealerweise direkt vor der Außengrenze des Wirbelkörpers.

In einer Evaluierung konnte ein durchschnittlicher DSC von 81,88% ermittelt werden (71,64% - 86,69%). Die rechenaufwendigsten Parametereinstellungen hatten eine Terminierungszeit von 19,1 Sekunden zur Folge (2,1 GHz, 4 GB RAM).

Acknowledgements

First, I like to take this opportunity to thank Prof. Dr. Bernd Freisleben and Dr. Dr. Jan Egger for the trust they placed in me and for their great supervision. I am especially grateful that my questions were always answered so promptly and that they showed so much interest in my work, which was a great encouragement to me.

Parts of this thesis were written at the Dept. of Radiology of Brigham And Women's Hospital of the Harvard Medical School, where I was introduced to many great scientists and where I was given the opportunity to make use of the excellent research facilities at the Surgical Planning Laboratory. I would like to thank Drs. Jan Egger and Tina Kapur for the invitation and the warm welcome as well as their great support throughout my stay in Boston.

Last but not least, I would like to thank my family and friends. I am especially indebted to my parents and my brother for their ever-lasting encouragement and on-going support.

Robert Schwarzenberg

Contents

1 Introduction

In this thesis, a new graph-based vertebral body segmentation approach is presented. In the first chapter *Medical Background*, the relative appearance of vertebral bodies in MRI data is discussed, thereby introducing the reader to the underlying framework conditions for vertebral segmentation. Furthermore, the section also deals with selected diseases of the spine that require a preoperative evaluation of a surgical patient's MR image(s). Hereby, the reader is introduced to potential fields of application of the new approach.

In this thesis, there will be an emphasis on time complexity since automatic vertebral body segmentation aims to decrease the time physicians spend on the preoperative evaluation of surgical patients. After the reader has been introduced to underlying mathematical concepts such as graphs and flow networks, external subroutines that the new algorithm deploys are presented, concluding with a discussion about their theoretical and experimental running times.

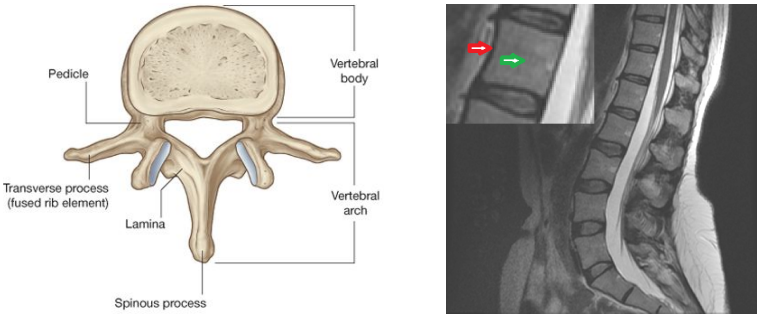
A C++ implementation of the novel approach was tested within the medical image processing platform MeVisLab [1]. The main features of the platform are presented in the section *MeVisLab* and the results of the evaluation are listed and discussed in the concluding paragraphs of this thesis. A detailed presentation of the actual algorithm can be found in the section *CubeCut* which also exemplifies how the new approach tackles typical problems in the field of vertebra segmentation.

2 Medical Background

The following section introduces the reader to the relative appearance of vertebral bodies in MR images[1]. Furthermore, the last paragraphs are concerned with selected diseases of the spine as well as corresponding preoperative measures, introducing the reader to potential fields of application of vertebral body segmentation.

2.1 Vertebral Bodies in MR Images

Several features of the spinal anatomy can be distinguished by their different grey values in an MR image. In most T1 and T2-weighted image slices, normal adult vertebral body bone marrow can be differentiated from the outer boundaries of the vertebral body by a homogeneously lighter grey value [2]. This is because the outer, compact cortical bone, which coats the vertebral body, results in a much darker color/lower grey value than the cancellous, spongy inner part.



(a) (b)

Figure 1: Vertebral anatomy. (a) illustrates the anatomy of a vertebra from a coronal view (adopted from [3]). (b) shows a sagittal T2-weighted MRI slice (for *coronal* and *sagittal* see Figure 5). The green arrow in the enlargement points to an area inside the vertebral body whereas the red arrow points to the cortical bone, the outer boundary.

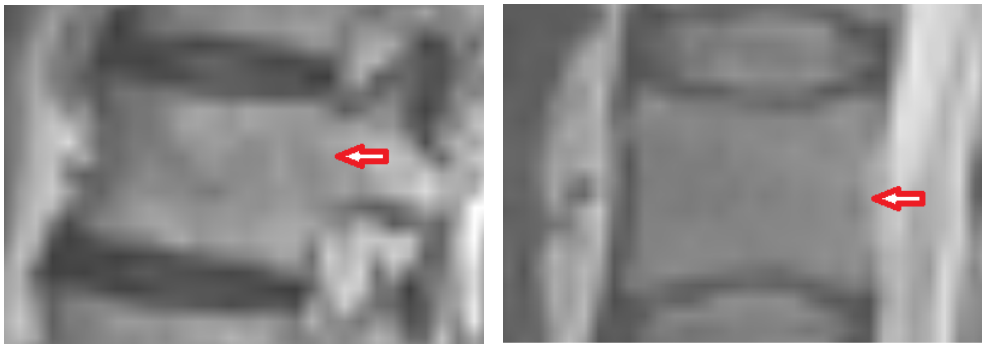
[1]For a thorough explanation of the physics and concepts of MR imaging, the reader is referred elsewhere.

Thus, the grey-value difference between a voxel in the vertebral body and a voxel on the outer boundaries (e.g. cortical bone) is higher than the difference between two voxels inside the vertebral body. This, however, does not apply to slices that depict the pedicles.

Figure 1 clearly shows that the pedicles of the vertebral arch are not considered part of the vertebral body. Nevertheless, since they are connected to the vertebral body, they belong to its outer boundaries. However, unlike the cortical bone, they define a weak, homogeneous object-background transition region.

Furthermore, in Figure 2 (a), instead of the cortical bone, the cerebrospinal fluid (CSF, surrounding the red arrow), which causes the high grey value of the spinal canal in T2-weighted images [2], defines parts of the outer boundary of the vertebral body. This is due to noise and signal distortion which results in an overlapping.

One can therefore conclude that because of signal distortion and noise, anatomical structures like the pedicles as well as occasional voxel outliers, the vertebral body cannot be defined by sharp boundaries in all MR image slices[2].



(a) (b)

Figure 2: Object-background transition regions (red arrows). (a) shows a homogenous object-background transition. In (b), the spinal canal (CSF) makes up parts of the vertebral body's outer boundaries.

[2]In what follows, the outer boundaries are classed with the background voxels.

2.2 Potential Fields of Application

Lumbar stenosis (LS), a narrowing of any part of the lumbar spinal canal with encroachment on the neural structures by surrounding bone and soft tissue [4, 5], is the most frequent reason for surgery in patients over 65 years of age [4]. While MR imaging is considered particularly purposive for the visualization of the soft tissue, X-ray computer tomography (CT) is seen as the method of choice for evaluating bone anatomy [4]. CT, however, exposes the patient to carcinogenic radiation while the magnetic field in MR imaging is harmless.

Sometimes, degenerative spondylolisthesis, an asymptomatic slipping forward of one lumbar vertebra on another with an intact neural arch, can be linked to LS [5]. Similar to LS, degenerative spondylolisthesis primarily occurs in elderly patients and a combination of MRI and CT is also applied for preoperative evaluations in this case. Note that a shift towards a more frequent application of MRI, even for morphological evaluations of the bone structure, would result in less radiation exposure [6].

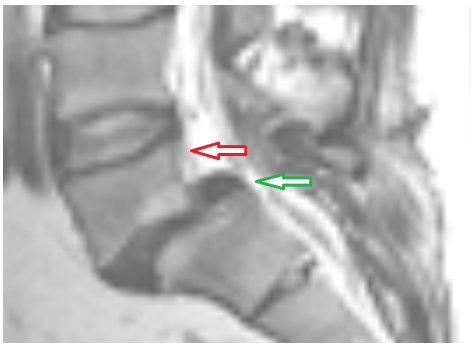


Figure 3: T2-weighted MR image showing a degenerative spondylolisthesis (red arrow) and a lumbar stenosis (green arrow).

3 MeVisLab

MeVisLab[3] is a platform allowing medical image processing and visualization. It provides basic and advanced algorithms as well as an interface for new C++ implementations.

Networks The platform is based on a modular pipelining principle and offers a GUI that allows the user to intuitively assemble a network of existing C++ modules, connecting them by "pipes". The first layer of such a network is typically a module that loads a medical image, e.g. `ImageLoad`, in the .dcm or .tif format, for instance (see Figure 4). `ImageLoad` provides an output field through which it propagates subimages of the loaded image towards an image input field of a connected module (e.g. `simpleAdd` in Figure 4).

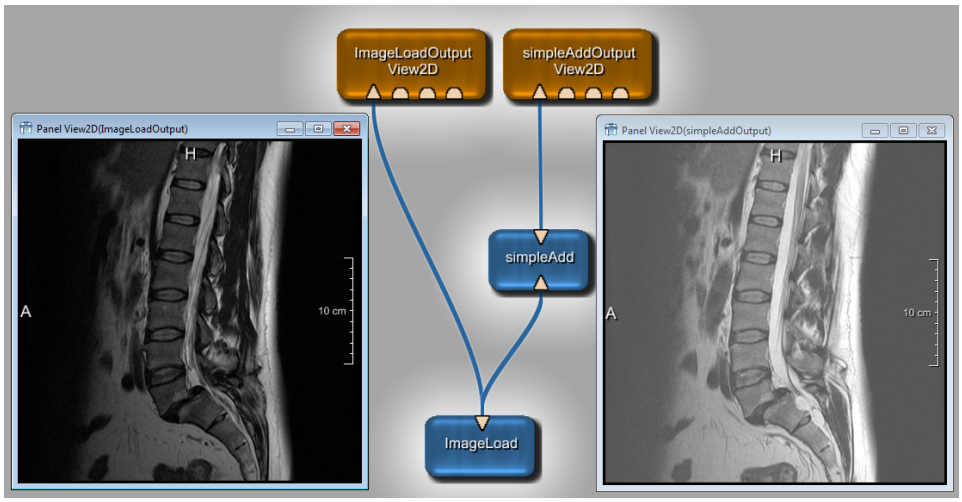


Figure 4: Screenshot of an example network in the MeVisLab workspace. `ImageLoad` loads an image and propagates it towards the image input fields (triangular-shaped) of modules in the next layer (here: `View2D` and `simpleAdd`). The `View2D` panels on the right and the left side visualize 2D slices of the input image. `simpleAdd` adds 300 to each voxel value of the input image (see Listing 1 and panel on the right side).

[3]The section at hand is solely based on the MeVisLab reference manual [1].

Image Processing Concepts It is up to the developer of a new ML (**M**evis**L**ab) module to decide on an image access strategy. MeVisLab supports paging, analogously to the concept of paging in memory management, as well as global image access.

The page-based concepts, such as the random accessing of voxels, should be applied if only local changes or information are needed. For algorithms that have to access the close environment of a voxel, MeVisLab provides a module that supports the so-called kernel-based approach which allows for a defined range around a given voxel to be addressed. A next possible step is the simultaneous loading of a complete image. In terms of memory management efficiency, the page-based approaches are most efficient while the full image loading is most costly.

Coordinate Systems A module providing an image input field, accepting the output of `ImageLoad`, for example, can address a position in the image using world coordinates as well as voxel coordinates. The *voxel coordinates* are continuous $[0...x, 0...y, 0...z]$, dependent on voxel spacing and they are often non-isotropic, which means that the voxels have different properties in different directions/perspectives. *World coordinates* on the other hand, measure distances and angles of one or more objects from a global perspective. It is possible to map voxel coordinates onto world coordinates (and reverse) by applying affine transformations (translation, rotation, scaling and shearing). The required orientation matrix is included in the image data.

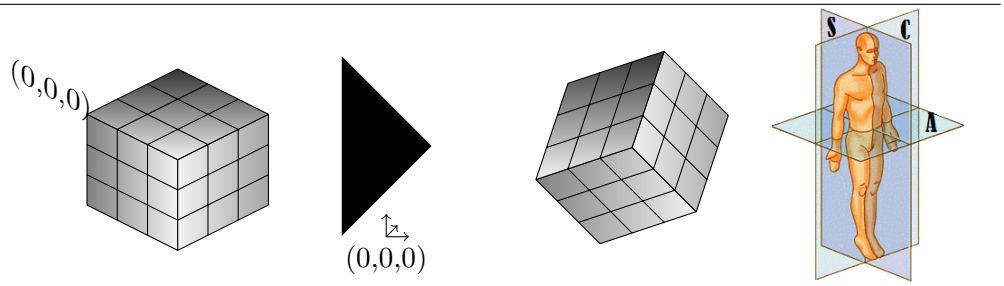


Figure 5: Mapping of voxel coordinates onto world coordinates. In medical imaging, the body planes on the right side (depiction adopted from [1]) serve as reference frames: S for *sagittal*, C for *coronal* and A for *axial*.

Example implementation The `simpleAdd` module in Figure 4 does not belong to the standard library. It has been developed using a "project wizard" that is part of the developing environment of MeVisLab. The wizard automatically generates an interface which can be enhanced by the developer. Part of the interface is the method `calculateOutputSubimage` (see Listing 1) that allows a page-based/voxel-based access of the input image as well as a page-based/voxel-based computing of the output image.

Listing 1 is an exact copy of the auto-generated method except for line 38 where an addend (300) has been appended. Line 38 results in a lightening of the image as can be seen in Figure 4. Without the extension, the module would simply pass the input image.

Listing 1: Page-based/voxel-based approach

```
1 //----------------------------------------------------------------------------
2 //! Template for type specific page calculation. Called by
3 //! ML_CALCULATEOUTPUTSUBIMAGE_NUM_INPUTS_1_CPP(simpleAdd);
4 //----------------------------------------------------------------------------
5 template <typename T>
6 void simpleAdd::calculateOutputSubImage(TSubImage<T>* outputSubImage,
7                                         int outputIndex,
8                                         TSubImage<T>* inputSubImage0
9                                        )
10 {
11   ML_TRACE_IN("template␣<typename␣T>␣simpleAdd::calculateOutputSubImage()");
12
13   // Compute subimage of output image outputIndex from input subimages.
14
15   //Clamp box of output image against image extent to avoid that unused areas are processed.
16
17   const SubImageBox validOutBox = outputSubImage->getValidRegion();
18
19   // Process all voxels of the valid region of the output page.
20   ImageVector p;
21   for (p.u=validOutBox.v1.u;  p.u<=validOutBox.v2.u;  ++p.u) {
22     for (p.t=validOutBox.v1.t;  p.t<=validOutBox.v2.t;  ++p.t) {
23       for (p.c=validOutBox.v1.c;  p.c<=validOutBox.v2.c;  ++p.c) {
24         for (p.z=validOutBox.v1.z;  p.z<=validOutBox.v2.z;  ++p.z) {
25           for (p.y=validOutBox.v1.y;  p.y<=validOutBox.v2.y;  ++p.y) {
26
27             p.x = validOutBox.v1.x;
28             // Get pointers to row starts of input and output subimages.
29             const T* inVoxel0 = inputSubImage0->getImagePointer(p);
30
31             T*  outVoxel = outputSubImage->getImagePointer(p);
32
33             const MLint rowEnd   = validOutBox.v2.x;
34
35             // Process all row voxels.
36             for (; p.x <= rowEnd;  ++p.x, ++outVoxel, ++inVoxel0)
37             {
38               *outVoxel = *inVoxel0 + 300;
39             }
40           }
41         }
42       }
43     }
44   }
45 }
```

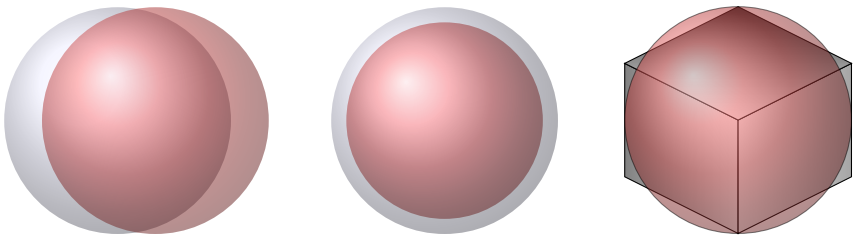
4 Preliminaries

4.1 Dice Similarity Coefficient

The Dice Similarity Coefficient (DSC) has been demonstrated to be an adequate indicator of medical image segmentation accuracy [7]. Consequently, it can also be used to detect failed segmentations [8]. It is defined as two times the cardinality of the intersection of a set A and a set B divided by the sum of the sets' cardinalities [9]:

$$\frac{2|(A \cap B)|}{|A| + |B|}$$

Thus, if the the DSC equals one, A equals B. A DSC equaling zero implies that A and B are disjoint. However, the DSC only takes the amount of overlaps into account, which is why it does not always allow us to make reliable predictions about the segmentation result's shape and/or position. Therefore, it can not be thought of as an absolute measure of segmentation accuracy.



(a) (b) (c)

Figure 6: Three segmentation results (set A, red shapes), objects (set B, grey shapes) and background (white). The resulting segmentation in (a) maintained shape and size of the original object but the center has been transformed. The segmentation result in (b) has the same center and shape as the original object but it is different in size. (c) shows a segmentation which did not maintain the shape of the original object but shares its center. Even though the differences between the DSCs in (a)-(c) are negligible, it has still to be decided which of the above segmentations can be considered successful.

4.2 Graphs

A *graph* G [4] is an ordered triple $(V(G), E(G), \psi_G)$. $V(G)$ and $E(G)$ are sets of *vertices* and *edges* and

$$\psi_G : E(G) \to V(G) \times V(G)$$

is an *incidence function* [11, 12]. ψ_G can also be defined by the two functions

$$o, t : E(G) \to V(G)$$

by

$$\psi_G(e) := (o(e), t(e)), e \in E(G)$$

[12]. With the order reversed, ψ_G also defines o and t by

$$(o(e), t(e)) := \psi_G(e)$$

[12]. $o(e)$ is referred to as the *origin* of e, whereas $t(e)$ refers to the *tail* of e [12]. If $v \in V(G)$ and $o(e) = v$ or $t(e) = v$, then v and e are *incident*, as well as $e, e' \in E(G)$, if e and e' share a vertex [12]. Two vertices $v, v' \in V(G)$ are *adjacent* if they are both incident with an edge e [11]. A *loopless* graph is one that has no edge e with $o(e) = t(e)$ [12]. If $v(G)$ and $e(G)$ denote the number of the graph's vertices and edges, $v(G)$ then is referred to as its *order*, while $e(G)$ describes its *size* [11]. G is *finite*, if $v(G), e(G) < \infty$ [11].

4.2.1 Simple Graphs

A graph $G = (V(G), E(G), \psi_G)$ is called *simple* if ψ_G is injective [12]. Thus, for all $v, v' \in V(G)$ at most one edge (v, v') can be assumed. If ψ_G is not injective, G is called a *multi graph* [12]. If G is simple, then $E(G)$ can be regarded as a subset of $V(G) \times V(G)$, which allows one to denote $\psi_G(E(G))$ by $E(G)$ since now $\psi_G(e) \neq \psi_G(e')$ if $e \neq e'$ for all $e, e' \in E(G)$ [12].

[4] In the subject literature, there are several slightly different definitions of graphs and the key terms associated with them. For instance, Dieter Jungnickel describes a graph as "an ordered pair (V, E) consisting of a finite set $V \neq \emptyset$ and a set E of two-element subsets of V"[10]. Thus, this definition excludes multi graphs, while others do not. Taking several sources into account in this chapter, a holistic approach is presented because it seems to be the most convenient for the further discussions.

Consequently, from now on, if a simple graph G is discussed, (v, v') denotes the edge e with $\psi_G(e) = (v, v')$ and one can simply write $G = (V(G), E(G))$ [12].

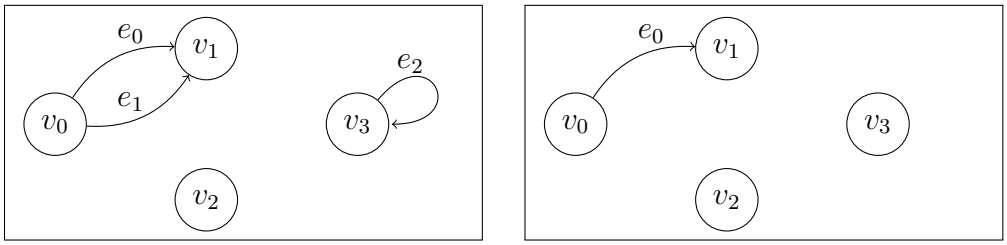


Figure 7: A multi graph (a) and a loopless, simple graph (b).[5]

Simple directed graphs

A simple, *directed* graph is an ordered pair $(V(G), E(G))$ where

$$E(G) \subseteq V(G) \times V(G)$$

(see Figure 7(b))[12].

Simple undirected graphs

A simple, *undirected* graph is a simple, directed graph $(E(G), V(G))$ which meets the following condition:

$$(v, v') \in E(G) \Leftrightarrow (v', v) \in E(G)$$

(see Figure 8(a))[12].

4.2.2 Paths

Let $G = (V(G), E(G), \psi_G)$ be a graph. A *path* P with

$$P = v_h, e_h, v_{h+1}, e_{h+1}, ..., e_{j-1}, v_j$$

[5]There are multiple ways to represent a graph correctly [11]. In general, the (relative) positioning of the vertices is arbitrary. Adjacent vertices are connected by arrows. In the case of an undirected graph, one usually draws an arc instead of two arrows (see Figure 8(a)). Here again, the shapes of the arcs and arrows are normally meaningless.

is an alternating (except in the case of a cyclic path) array of vertices $v_i \in V(G)$ and edges $e_i \in E(G)$, where $\psi_G(e_i) = (v_i, v_{i+1})$ for $h \leq i < j$ [13]. In the case of a simple graph G, P can be characterized by an array of just vertices: $v_h, ..., v_j$ [13]. P then is said to be *simple* [13].

4.2.3 Connectedness

Let $G = (V(G), E(G), \psi_G)$ be a graph. $v_h, v_j \in V(G)$ are *connected* if there exists a path $P = v_h, e_h, ..., e_{j-1}, v_j$ [13]. Furthermore, by convention, there is always a connection from v_h to v_h [13]. If G is a simple, directed graph and if for all $v_h, v_j \in V(G)$ there exists a path P from v_h to v_j or from v_j to v_h, then G is *connected* [14]. Otherwise it is *disconnected*. G is *strongly connected* if for all $v_h, v_j \in V(G)$ there exists a path from v_h to v_j and a path from v_j to v_h [14].

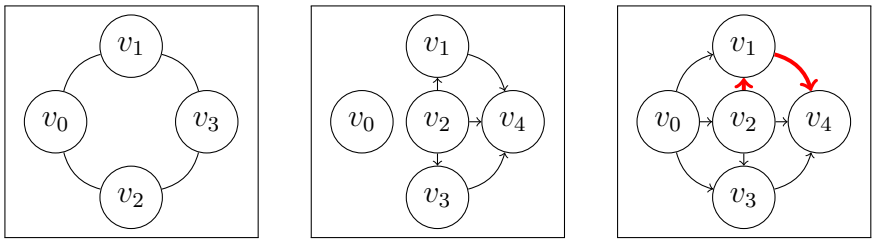


Figure 8: A strongly connected, undirected graph (a), a disconnected graph (b), and a connected graph (c). A path v_2, v_1, v_4 is marked in red.

4.2.4 Cuts

Let $G = (V(G), E(G))$ be a simple, directed graph. If $C = (V, \overline{V})$ is a partition of $V(G)$, so that $V \cup \overline{V} = V(G)$ and $V \cap \overline{V} = \emptyset$, then

$$\{e \in E(G) | (o(e) \in V \wedge t(e) \in \overline{V}) \vee (t(e) \in \overline{V} \wedge o(e) \in V)\}$$

is a *cut-set* and C is a *cut* [15].

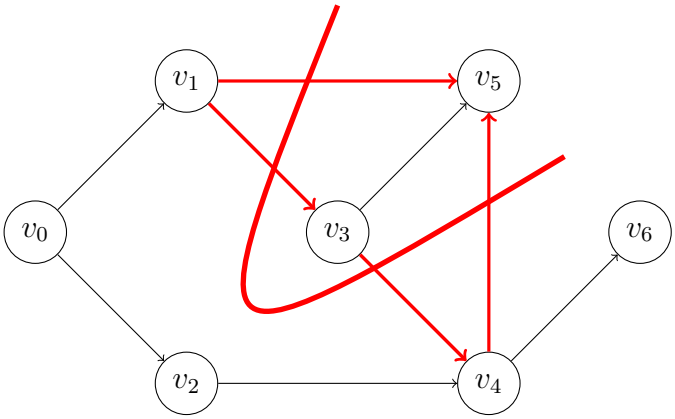


Figure 9: A graph-cut $C = (\{v_0, v_1, v_2, v_4, v_6\}, \{v_3, v_5\})$ and the corresponding cut-set $\{(v_1, v_3), (v_1, v_5), (v_3, v_4), (v_4, v_5)\}$.

4.3 Networks and Flows

4.3.1 Flow Networks

A *network* $N = (G, c, s, t)$ consists of a finite, simple, loopless, directed, and connected graph G = (V(G),E(G)) and a *capacity function*

$$c : E(G) \to \mathbb{R}_{\geq 0},$$

which assigns a non-negative real number to each edge in the edge set[16, 17]. If $(v, v') \notin E(G)$, one assumes that $c(v, v') = 0$ [16]. Furthermore, a network consists of (at least) [6] two distinguished vertices s and t, the *source* and the *sink*. A *flow* is a real-valued function

$$f : V(G) \times V(G) \to \mathbb{R},$$

that satisfies the given conditions:

$$Capacity \quad constraint : \quad \forall v, v' \in V(G) : f(v, v') \leq c(v, v'),$$

[6] Henceforth a single-source, single-sink network is assumed and, for convenience, it is implied that all vertices lie on a path $s, ..., t$.

$$Skew \quad symmetry: \quad \forall v, v^{'} \in V(G) : f(v, v^{'}) = -f(v^{'}, v),$$

$$Flow \quad conservation : \forall v \in V(G) \backslash \{s, t\} : \sum_{v^{'} \in V(G)} f(v, v^{'}) = 0$$

[16, 17].

The *capacity constraint* simply limits the flow from one vertex to another to a value less than or equal to the capacity of the common edge. The *skew symmetry* determines that the two-way flow between two vertices equals zero. The *flow conservation* equation says that none of the *intermediate* vertices, that is all vertices except for the source and the sink, "create" or "absorb" flow. Observe that if neither $(v, v^{'}) \in E(G)$ nor $(v^{'}, v) \in E(G)$, then $f(v, v^{'}) = 0$ and $f(v^{'}, v) = 0$ (because $c(v, v^{'}) = 0$ and $c(v^{'}, v) = 0$).

Only considering the total positive flow entering and leaving an intermediate vertex $v \in V(G) \backslash \{s, t\}$, due to the conservation property, it becomes apparent that the "flow in equals the flow out:"

$$\sum_{\substack{v^{'} \in V(G) \\ f(v^{'}, v) > 0}} f(v^{'}, v) = \sum_{\substack{v^{'} \in V(G) \\ f(v, v^{'}) > 0}} f(v, v^{'}) \tag{1}$$

[16]. Thus, here again, no flow gets "lost" or "accumulates" in the intermediate vertices.

The *value* of a flow f, denoted by $w(f)$, is the *total net flow* at the source [16, 10]. The total net flow at any vertex is defined as the positive flow in minus the positive flow out [16]. Note that because of (1), the total net flow at the sink equals $-w(f)$.

4.3.2 Residual Networks

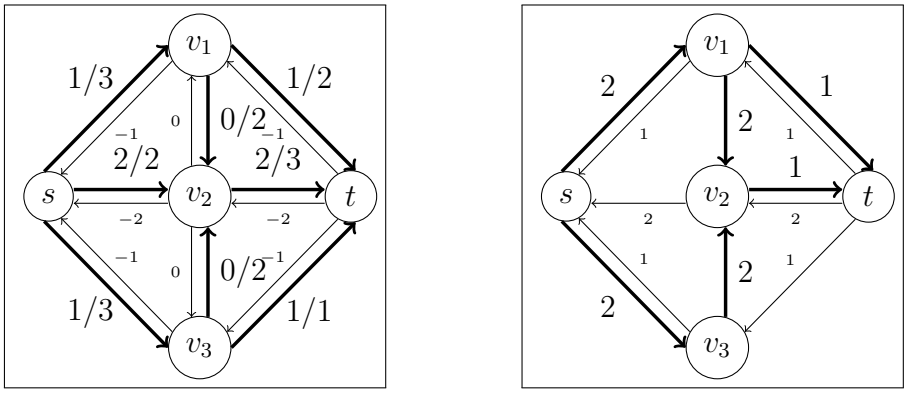
$N_f = ((V(G), E_f(G)), c_f, s, t)$ is the *residual* network of a network $N = ((V(G), E(G), c, s, t))$ and a flow f if

$$E_f(G) = \{(v, v^{'}) \in V(G) \times V(G) | c_f(v, v^{'}) > 0\}$$

and

$$c_f(v, v^{'}) = c(v, v^{'}) - f(v, v^{'})$$

[16]. Note that $c_f(\cdot)$ is always positive for all edges in the residual network [16].



(a) (b)

Figure 10: A flow network (a) and the induced residual network(b). Highlighted in bold are the edges which are actually in the set of edges. They are labeled with the positive flow/capacity ratio. (b) shows the induced residual network of (a).

4.3.3 Max-Flow Min-Cut Theorem

Given a network N, an *s-t-cut* is a cut $C = (S, T)$, so that $s \in S$ and $t \in T$. The capacity $|C|$ of such an s-t-cut is the sum of the capacities of all *forward arcs*, that is all edges $e = (v, v') \in E(G)$ for which $o(e) \in S$ and $t(e) \in T$:

$$|(S,T)| = \sum_{\substack{(v,v')\in E(G) \\ v\in S \\ v'\in T}} c(v, v')$$

[18]. If for an s-t-cut C_{min}, there exists no other s-t-cut C' so that $|C'| < |C_{min}|$, then C_{min} is a *minimal cut* [19].

In 1962 L. R. Ford and D. R. Fulkerson stated and proved that "for any network the maximal flow value from s to t is equal to the minimal cut capacity of all cuts separating s and t" [17]. Thus, $w(f)$ is of maximal value, if and only if the flow at the forward arcs in the cut-set of any minimal cut equals these arcs' capacities. Consequently, given a maximal flow $f*$, a minimal s-t-cut can be found if one tags all edges at which the flow equals the capacity and then identifies an s-t-cut C_{min}, whose cut-set only contains tagged edges and where $w(f*) = |C_{min}|$.

5 Max-Flow Min-Cut Algorithms

5.1 Ford-Fulkerson Algorithm

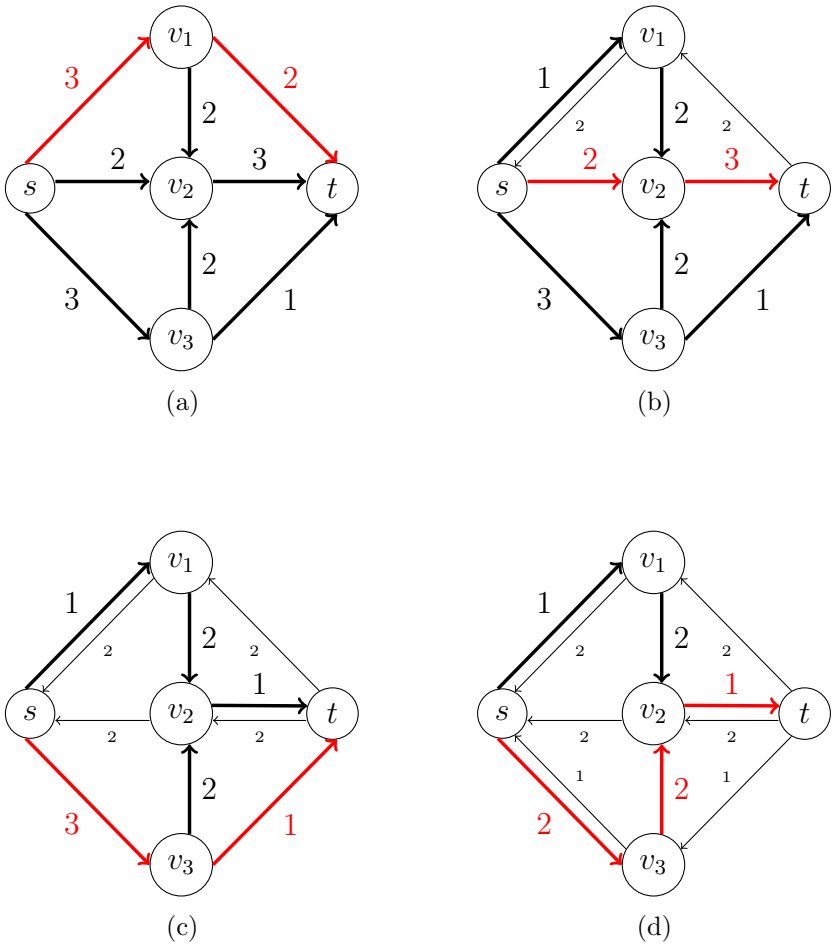
The maximum flow $f*$ in a network N can be determined by the *Ford-Fulkerson* algorithm. The algorithm works with augmenting paths. An *augmenting path* is a path from s to t in the residual network N_f [16]. By definition, all edges $(v, v') \in E_f(G)$ are induced by those edges in $E(G)$ at which $c(v, v') > f(v, v')$ (compare Ch. 4.3.2) and thus, if there exists a path $P = s, ..., t$ in N_f, the flow along P can be augmented by $min\{c_f(v, v')|(v, v') \in P\}$, without violating the capacity constraint [16].

The Ford-Fulkerson algorithm repeatedly searches for an augmenting path; if the search is successful, it then maximizes the flow as described above. When no path from s to t can be found in N_f (anymore), then $w(f)$ is of maximal value [17].

```
0   void   FF()
1   ∀(v, v') ∈ N
2           f(v, v') ← 0
3           f(v', v) ← 0
4   N_f ← residual()
5   while  (∃P = s, ..., t  in  N_f)
6           c_f(P) ← min{c_f(v, v')|(v, v') ∈ P}
7           ∀(v, v') ∈ P
8                   f(v, v') ← f(v, v') + c_f(P)
9                   f(v', v) ← −f(v, v')
10          N_f ← residual()
```

Figure 11: Pseudo code of the Ford-Fulkerson algorithm. The algorithm mutates a flow f. It is assumed that f and the network N are fields in the global scope. *residual* returns the residual network N_f, induced by N and f.

The running time of the Ford-Fulkerson algorithm depends on the search strategy used to find an augmenting path (line 5). The shortest path can be found by deploying a breadth-first search. If the search is taken out of the calculation and if $c : E(G) \to \mathbb{N}$ can be assumed, the algorithm then terminates in time $O(e(G)|C_{min}|)$, for lines $1 - 3$ take $e(G)$ steps while the condition in line 5 becomes false after at most $|C_{min}|$ steps [16]. Figure 12 illustrates the Ford-Fulkerson algorithm starting at line 5 in the pseudo code (see Figure 11).



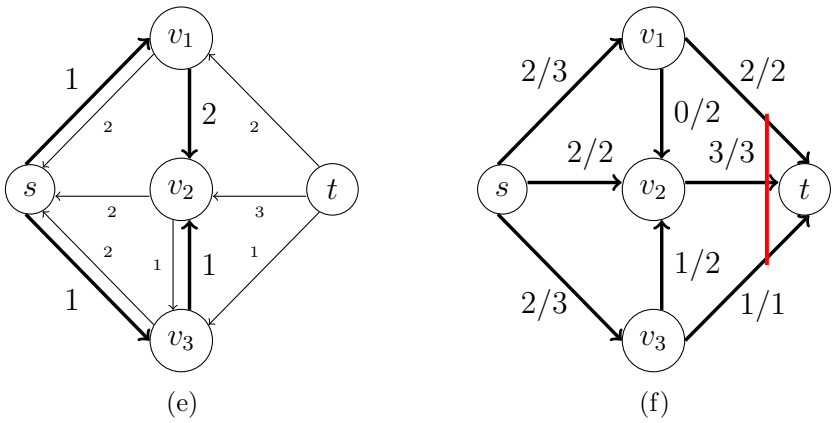


Figure 12: Illustration of the Ford-Fulkerson algorithm. (a)-(e) show the residual networks resulting from the computations in line 4 ((a)) and line 10 ((b)-(e)) (see Figure 11). (f) shows the resulting network and the minimal cut (red) after termination.

5.2 Goldberg-Tarjan Algorithm

Until 1988, the year that A.V. Goldberg and R.E. Tarjan introduced a new approach to solve the maximum flow problem, all previous, efficient algorithms worked with augmenting paths, adopting Ford and Fulkerson's idea as described above [20]. As a matter of fact, today, one can link most of the algorithms to either the Ford-Fulkerson algorithm or to the *Goldberg-Tarjan* algorithm (also referred to as *push-relabel*), which is why they are considered standard algorithms in this field [21].

The following paragraphs outline the push-relabel approach as described in A.V. Goldberg's and R.E. Tarjan's 1988 paper *"A new approach to the maximum-flow problem,"* which also provides a detailed proof of correctness [20]. However, before discussing the algorithm, a few new features have to be introduced.

5.2.1 New Features

Given a Network N and a flow f, the Goldberg-Tarjan algorithm can intermediately violate the flow conservation constraint by allowing vertices to

accumulate flow so that

$$\forall v \in V(G) : \sum_{\substack{v' \in V(G) \\ f_p(v',v)>0}} f_p(v', v) \geq \sum_{\substack{v' \in V(G) \\ f_p(v,v')>0}} f_p(v, v').$$

This is because the algorithm first generates a so-called *preflow* f_p, which satisfies the capacity constraint and the skew symmetry as well as a third invariant, which determines that the preflow out must not exceed the preflow in:

$$\textit{Nonnegativity} \quad \textit{constraint} : \quad \forall v' \in V(G) \backslash \{s\} : \sum_{v \in V(G)} f_p(v, v') \geq 0.$$

On the other hand, if the preflow into a vertex $v \in V(G)$ exceeds the preflow out of it, the *nonnegativity constraint* will hold true. In this case, v accumulates preflow units and due to the skew symmetry constraint, the resulting *excess* $e_x(\cdot)$ can simply be described by the total preflow into v:

$$e_x(v) = \sum_{v' \in V(G)} f_p(v', v).$$

Observe that the *nonnegativity constraint* is a weakening of the flow conservation constraint and that if

$$\forall v \in V(G) \backslash \{s, t\} : e_x(v) = 0,$$

then f_p is a flow.

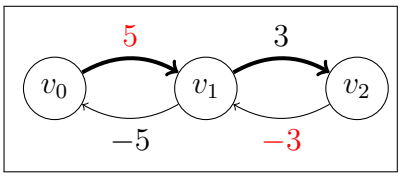


Figure 13: Illustration of a positive flow excess: $e_x(v_1) = 2$.

The authors also introduced a function

$$l : V(G) \to \mathbb{N}_0 \cup \{\infty\},$$

where $l(s)(= |V(G)|)$ and $l(t)(= 0)$ are constants and for which the following invariant always holds true:

$$\forall (v, v') \in E_{f_p}(G) : l(v) \leq l(v') + 1.$$

Furthermore, for all vertices other than the source or the sink, l is defined dynamically during runtime in such a way that at any time $l(v)$ is to be interpreted in the following manner:

- $l(v) < l(s)$ means that $l(v)$ is a lower bound of the length of the shortest path $P = v, ..., t$ and
- $l(v) \geq l(s)$ means that $l(v) - l(s)$ is a lower bound of the length of the shortest path $P = v, ..., s$.

Thus, l allows us to estimate the distance from a vertex to the source or to the sink. In the following, a vertex $v \in V(G) \backslash \{s, t\}$ is said to be *active* if $l(v) < \infty$ and if $e_x(v) > 0$. There are two basic operations the algorithm repeatedly applies if the respective invariants hold:

- If v is active and if $(v, v') \in E_{f_p}(G)$ and if $l(v) = l(v') + 1$, then a *push* is allowed which means that the preflow on (v, v') is increased by $\delta = min\{e_x(v), c_f(v, v')\}$.
- If v is active and if $\forall v' \in V(G) : (v, v') \in E_{f_p}(G) \Rightarrow l(v) \leq l(v')$, then a *relabeling* is allowed, which means that $min\{l(v') + 1 | (v, v') \in E_{f_p}\}$ is assigned to $l(v)$ if $\{(v, v') | (v, v') \in E_{f_p}\} \neq \emptyset$ and ∞ is assigned otherwise.

Observe that a push is only allowed "downwards" from a higher l-level towards a lower level. Furthermore, note that by pushing, $e_x(v)$ is decreased by δ, while $e_x(v')$ is increased proportionally and that the capacity constraint is never violated. Moreover, due to skew symmetry, which a preflow satisfies by definition, the preflow on (v', v) has to be decreased by δ at the same time. Thus, after a push, the excess of v has been propagated towards v'.

A relabeling or "lifting" is allowed if the excess of an active vertex v cannot be pushed. After "lifting" v, a push is possible again.

5.2.2 The Algorithm

During an initialization phase, the algorithm increases the flow on all residual edges $(s, v) \in E_{f_p}(G)$ to their capacity. Thus, after preprocessing, all adjacent vertices of the source s are active. The algorithm then tries to propagate these excesses forward towards the sink t. In order to achieve this, it repeatedly examines the set of active vertices (in the following $V(G)_{act}$) and the set of residual edges.

If there exists a vertex $v \in V(G)_{act}$ and a residual edge $(v, v') \in E_{f_p}(G)$, where v' is closer bound to t, the algorithm then pushes v's excess towards v'. Note that if $c_f(v, v') < e_x(v)$, then only a fraction of the excess can be pushed. The rest remains and thus, v stays active.

However, after initialization, $l(v) = 0$ for all vertices except for the source so that no excess is allowed to be pushed, yet. If the set of active vertices is not empty but a push of excesses is not allowed (because there exists no "inclining," residual edge), the active vertices are then "lifted" or relabeled. After relabeling, a push is (again) possible.

The algorithm repeats this procedure of pushing and relabeling until eventually the maximum amount of preflow, restricted by the capacity constraint, reaches the sink. In order to convert the preflow into a flow, the algorithm now propagates any remaining excesses back towards the source. To do so, it deploys the same procedure as before. It terminates when no active vertices are left.

---

0 *bool* **push**()
1 *if* $\exists \mathbf{v} \in \mathbf{V(G)_{act}}$ *and* *if* $\exists (\mathbf{v}, \mathbf{v'}) \in \mathbf{E_{f_p}(G)}$ *and* *if* $\mathbf{l(v)} = \mathbf{l(v')} + \mathbf{1}$
2 $\quad \delta \leftarrow \mathbf{min}\{\mathbf{e_x(v)}, \mathbf{c_f(v, v')}\}$
3 $\quad \mathbf{f_p(v, v')} \leftarrow \mathbf{f_p(v, v')} + \delta$
4 $\quad \mathbf{f_p(v', v)} \leftarrow \mathbf{f_p(v', v)} - \delta$
5 $\quad \mathbf{e_x(v)} \leftarrow \mathbf{e_x(v)} - \delta$
6 $\quad \mathbf{e_x(v)} \leftarrow \mathbf{e_x(v)} + \delta$
7 $\quad$ *return* **true**
8 *return* **false**

```
0  bool  relabel()
1  if  ∃v ∈ V(G)_act  and  if  ∀v' ∈ V(G) : (v, v') ∈ E_fp(G) ⇒ l(v) ≤ l(v')
2      if  {(v, v')|(v, v') ∈ E_fp(G)} ≠ ∅
3          l(v) ← min{d(v') + 1|(v, v') ∈ E_fp(G)}
4          return  true
5      else
6          l(v) ← ∞
7          return  true
8  return  false

0  void  push − relabel()
1  ∀(v, v') ∈ V(G) × V(G)
2      f_p(v, v') ← 0
3  ∀v ∈ V(G)
4      f_p(s, v) ← c(s, v)
5      f_p(v, s) ← −c(v, s)
6      e_x(v) ← f_p(s, v)
6      l(v) ← 0
1  l(s) ← |V(G)|
7  V(G)_act ← activeSet()
8  N_fp ← residual()
9  if  push()  or  relabel()
10     goto  7
```

Figure 14: Pseudo code of the generic push-relabel algorithm. *Generic* points to the fact that the active vertices and residual edges are consulted in an arbitrary order. *residual* returns the residual network N_{f_p}, induced by the network N and the preflow f_p. *activeSet* determines the subset of active vertices in $V(G)$ and returns it. The main method begins by initializing the preflow f_p, the distance mapping l and excess mapping e_x. It is assumed that all variables mentioned above lie in the global scope of the program.

The generic algorithm terminates after at most $O(v(G)^3)$. By using dynamic tree-structures, the runtime can be improved to $O(v(G)e(G)\log(v(G)^2/e(G)))$.

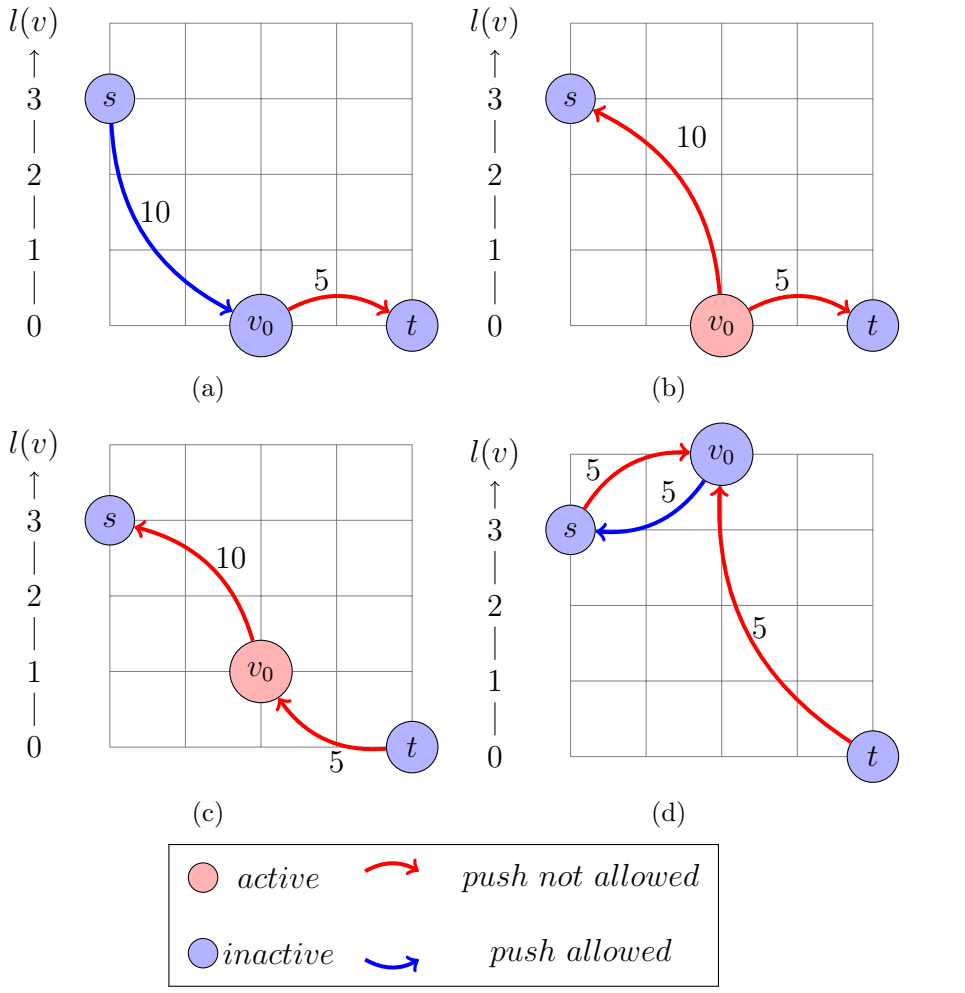


Figure 15: Illustration of the Goldberg-Tarjan algorithm. (a) shows a network N. (b) shows the residual network N_{f_p} after the initialization phase. (c) shows N_{f_p} after v has been relabeled and $e_x(v_0)$ has been pushed towards t. (d) shows the residual network after termination.

5.3 Boykov-Kolmogorov Algorithm

In 2004, Y. Boykov and V. Kolmogorov presented a min-cut/max-flow algorithm that frequently outperformed efficient standard algorithms in image processing (compare Ch. 5.4)[21] [7]. Interestingly, the theoretical time complexity of the new approach is worse than that of the algorithms with which it was compared. Adopting the ideas of Ford/Fulkerson, the algorithm makes use of augmenting paths:

Given a flow network N, the algorithm generates two non-overlapping trees (S and T), one of which has the source as its root (S) and another one which expands from the sink (T). Vertices that are not part of either of the two trees are referred to as *free*. A vertex in a tree's collection is either said to be *active* or *passive*, depending on whether or not all of its neighbors have been explored. A vertex explores its neighborhood by adopting all adjacent, free vertices with which it is connected via an incident, residual edge. Thus, the outer border of a tree consists of a set of active vertices that allow exploration and expansion, while the internal, passive vertices have already explored their neighborhood.

The acquiring of free, adjacent vertices (expansion of the trees) takes place when the algorithm is in the *growth stage*. Note that in the course of the expansion, some of the active vertices might become passive. The trees expand until one of their active vertices identifies a residual edge which connects it to the outer border of the other tree. In this case, an augmenting path has been detected.

Now, the algorithm enters the *augmenting stage* in which it augments the flow along the detected path. Afterwards, saturated edges are deleted from the lists of the two trees and as a consequence, some incident vertices might be disconnected. The authors refer to the disconnected vertices as *orphans*. It should also be noted that the deletion of the saturated edges and the expelling of the disconnected vertices may break up the trees into disjoint unions.

To rebuild joint unions, the algorithm now enters the *adoption stage* in which it tries to reconnect the orphans (and their children) to the tree to which they they belong. On this, it tries to find residual edges that connect active vertices with the orphans. All of the orphans that are not incident to such a residual edge are then declared free again and their children are declared orphans. The adoption stage is completed when there are no or-

[7] This section is based solely on [21].

phans left.

The algorithm repeats the three stages until the sets of active vertices are empty and the trees are disjoint. After termination, the flow is a maximal flow since the trees are separated by saturated edges which form a minimal cut.

Thus, one can conclude that the algorithm's good performance results from its search strategy: Instead of searching for a shortest augmenting path every new iteration by a breadth first search, the algorithm reuses the trees that are already set up.

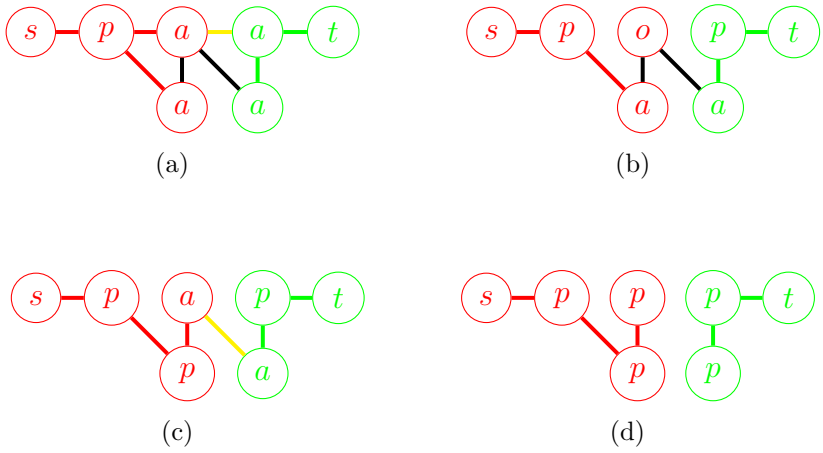


Figure 16: Illustration of the Boykov-Kolmogorov algorithm. (a) - (d) show residual networks at different stages of the algorithm. Red arcs and vertices belong to S while green arcs and vertices belong to T. Black edges haven't been explored, yet. (a) residual network after growth stage, (b) after augmenting stage, (c) after adoption stage, (d) after termination.

Figure 17 shows the pseudo code of the algorithm.

$$tree : V(G) \rightarrow \{S, T, \emptyset\}$$

is a function which maps a vertex to either one of the search trees S or T, or to $\emptyset$, if the vertex is free. If $tree(v) \neq \emptyset$, then v's parent is determined by *parent(v)*.

$$neighbor(v) = \{v' | (v, v') \in E_f\}$$

is a set of all adjacent, free vertices $v' \in V(G)$, that share a common, non-saturated edge with v. A is a list of the active vertices, O is a list of the orphans. Boykov and Kolmogorov proved that in the worst case, the algorithm terminates in time $O(e(G)v(G)^2|C_{min}|)$.

```
0  bool  BK()
1  S ← {s}
2  T ← {t}
3  A ← {s, t}
4  O ← ∅
5  while  true
6      P ← grow()
7      if  P == ∅
7          return
9      augment(P)
10     adopt()
```

```
0  P  grow()
1  while  ∃v ∈ A
2      ∀v′ ∈ neighbor(v)
3          if  tree(v′) == ∅
4              tree(v′) ← tree(v)
5              parent(v′) ← v
6              A ← A ∪ {v′}
7          if  tree(v′) ≠ ∅  and  tree(v) ≠ tree(v′)
8              return  P_{s,...t}
9      A ← A\{v}
10 return  ∅
```

```
0  void  augment(P)
1  augmentPath(P)
2  ∀(v, v′) ∈ P
3      if  c_f(v, v′) == f(v, v′)
4          if  tree(v) == tree(v′) == S
4                  parent(v′) ← ∅
6                  O ← O ∪ {v′}
5          if  tree(v) == tree(v′) == T
7                  parent(v) ← ∅
8                  O ← O ∪ {v}
```

```
0  void  adopt()
1  while  ∃v ∈ O
2      O ← O\{v}
3      process(v)
```

Figure 17: Pseudo code of the max-flow algorithm by Boykov and Kolmogorov. *augmentPath* augments the flow along a path P. *process(v)* tries to find a valid parent for a vertex v. A valid parent v' belongs to the same tree as v (S or T) and is connected to it via a non-saturated edge. If a new parent is found, *parent(v)* is updated. Otherwise *neighbor(v)* is consulted and all vertices of the same root as v are added to the active set A and all of $v's$ children $v' \in$ *neighborhood(v)* are added to O, then *parent(v')* is updated. Furthermore, *process(v)* then declares v free in a last step.

5.4 Experimental Comparison

Boykov and Kolmogorov compared the runtime of their algorithm with previous augmenting path and push-relabel style implementations [21]. The algorithms found minimal s-t-cuts in two-terminal networks consisting of (3D)

regular grid graphs with N_6 and N_{26} neighborhood systems.

In addition to (theoretically) less efficient approaches, an augmenting path-based algorithm, discovered by E. A. Dinic in 1970 [22] and a push-relabel algorithm, a fine tuned version of the Goldberg-Tarjan algorithm [20], were tested. The worst case complexities are listed in Table 1 and the experimental running times (1.4 GHz Pentium IV) can be found in tables 2 and 3 [21].

Table 1: Worst case complexities for the tested augmenting path (ap) based and push-relabel (pr) style algorithms [21].

algorithm	style	worst case complexity
Dinic	ap	$O(v(G)^2 e(G))$
Goldberg-Tarjan	pr	$O(v(G)^2 \sqrt{e(G)})$
Boykov-Kolmogorov	ap	$O(v(G)^2 e(G) \lvert C_{min} \rvert)$

Table 2: Running times ($127 \times 127 \times 12$ vertices) [21].

algorithm	runtime/seconds (N_6)	runtime/seconds (N_{26})
Dinic	20.16	39.13
Goldberg-Tarjan	1.38	2.44
Boykov-Kolmogorov	0.7	2.44

Table 3: Running times ($76 \times 399 \times 38$ vertices) [21].

algorithm	runtime/seconds (N_6)	runtime/seconds (N_{26})
Dinic	172.41	443.88
Goldberg-Tarjan	18.19	47.99
Boykov-Kolmogorov	13.22	90.64

Conclusion The results suggest that despite its theoretical worst case complexity, the Boykov-Kolmogorov algorithm is most efficient when it comes to the mincut computation in s-t-networks of low complexity, e.g. up to N_6 neighborhood systems.

6 CubeCut: Vertebral Body Segmentation

The new vertebral body segmentation algorithm presented here will be referred to as *CubeCut*. CubeCut extends a two-dimensional approach, previously introduced by Jan Egger et al., to a third dimension [23] and is strongly related to [24, 25, 26, 27].

The introductory paragraphs of this section first give a conceptual, abstract overview of the basic features and the behaviour of CubeCut. The introduction serves as a frame of reference for the more detailed discussion that follows at a later stage.

6.1 Abstract Overview

6.1.1 Labeling

Given a volumetric MR image P, CubeCut first selects a subset $P' \subseteq P$ of the image's voxels and in a last step it tags each voxel $p \in P'$ with either one of the labels L_s or L_t [21]:

$$t : P' \rightarrow \{L_s, L_t\}.$$

6.1.2 Penalties

The labeling of a voxel $p \in P'$ involves two penalties [21]:

- $D_p(t(p)) \in \mathbb{R}_{\geq 0}$ is the penalty for assigning the label $t(p)$ to p and
- $V_{p,p'}(t(p), t(p')) \in \mathbb{R}_{\geq 0}$ is the penalty for assigning $t(p)$ to p when $t(p')$ is the label of the voxel $p' \in P'$.

D describes a voxel's affinities to the labels L_s and L_t. For example, the higher $D_p(t(p) = L_t)$, the more p is affiliated with L_s. V on the other hand reflects a voxel's affiliation to another voxel. In practice, $V_{p,p'}(t(p), t(p'))$ is greater than zero only if $t(p) \neq t(p')$. Thus, V indirectly describes p's affiliation with p' by awarding a penalty for tagging the two voxels with different labels: The higher $V_{p,p'}$, the more p is affiliated with p'.

Nevertheless, note that $V_{p,p'}(t(p), t(p')) = 0$ for $t(p) \neq t(p')$ does not necessarily mean that the two voxels p and p' can be tagged differently without penalty costs. If, for instance, $V_{p,p''}(t(p), t(p'')) > 0$, for $t(p) \neq t(p'')$ and $p'' \in P'$ and if $V_{p'',p'}(t(p''), t(p')) > 0$ for $t(p'') \neq t(p')$, then the penalty cost for assigning different labels to p and p' is at least $min\{V_{p,p''}, V_{p'',p'}\}$.

6.1.3 Return Value

CubeCut tags the voxels in a way such that the overall penalty cost is minimized. The overall cost is described by (2). CubeCut thus returns the argument L which minimizes

$$E(L) = \sum_{\substack{p \in P' \\ t(p) \in L}} D_p(t(p)) + \sum_{\substack{p,p' \in P' \\ t(p),t(p') \in L}} V_{p,p'}(t(p), t(p')), \tag{2}$$

where $L = \{t(p) | p \in P'\}$ is a labeling of the subset P' [21, 28]. However, until now, the features above have been discussed detached from the context of vertebral segmentation. The next section will make the connection.

6.1.4 Object and Background Separation

CubeCut selects P' and implements D and V (on the basis of the predetermined subset P') in a way such that the returned labeling is to be interpreted in the following manner:

- CubeCut assumes all voxels $p \in P'$ for which $t(p) = L_s$ inside the vertebral body and
- all voxels $p \in P'$ for which $t(p) = L_t$ can be assumed outside the vertebral body.

Figure 18: Illustration of voxel labeling.

Hence, so far, one can conclude that in a first step CubeCut selects a subset of voxels and on the basis of this subset, implements two penalty functions which then determine a clustering of the subset into two disjoint units of voxels. One unit describes the vertebral body while the other describes the background (which may include other vertebrae). The following paragraphs will put this meta-view of the algorithm into concrete terms.

---

0 *labeling* **CubeCut**(**P**)
1 $\mathbf{P'} \leftarrow$ **generateP'**(**P**)
2 $(\mathbf{D}, \mathbf{V}) \leftarrow$ **implementDandV**($\mathbf{P'}$)
3 *return* **argmin**$[E, L]$

Figure 19: Pseudo code of CubeCut.

6.2 The Voxel Subset

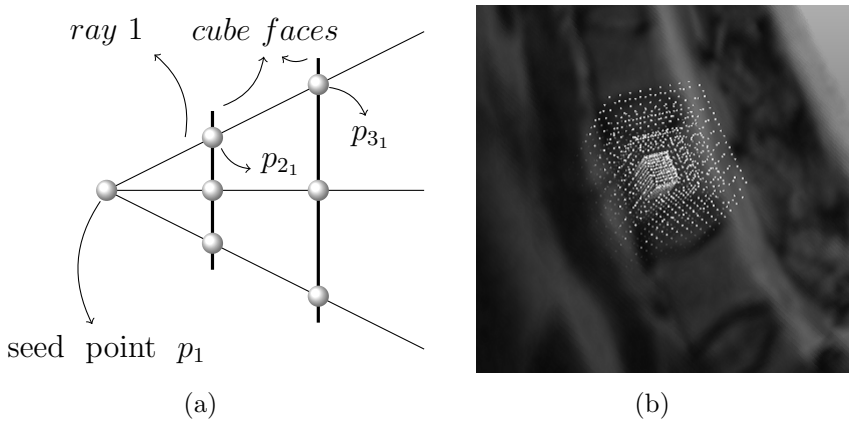
The voxels $p \in P'$ are distributed along n rays that expand from a user-defined seed point in the MR image. Each ray consists of k equidistantly spread voxels, where for all rays, the first voxel is always the user-defined seed point, so that $|P'| = n * (k-1) + 1$. As the seed point, the number and the length of the rays as well as the number of voxels per ray can be determined by the user. It is assumed that each ray exceeds the vertebral body. In the following, let $p_{i_r} \in P'$ denote a voxel on ray r, where $1 \leq r \leq n$ and where the voxel p_{i_r} is closer to the seed point (p_1 *or* p_{1_r}) than p_{j_r}, if $1 \leq i < j \leq k$ [8].

6.2.1 Cubic Distribution

The rays expand in a way such that all voxels of the same layer form a cube shape, so that if $i = j \neq 1$ and $m \neq n$, then the voxels p_{i_m} and p_{j_n} lie on the surface of one cube, which has the user-defined seed point as its center. Since

[8] If only one ray is being discussed, the indexing r might be omitted. Furthermore, from now on, k will always denote the number of voxels per ray and n the number of rays.

there are k voxels on each ray, there are $k-1$ different sized cubes for which p_1 is the center (see Figure 20 (b)). On a cube's face, the voxels are distributed equidistantly and the volumes of the cubes increase evenly. However, note that due to the theorem of intersecting lines, the distance between two voxels on a cube's face is less than the distance of the corresponding voxels on a bigger cube.



(a) (b)

Figure 20: Profile of two cube faces intersected by three rays (a) and a cubic voxel subset (b).

6.3 Implementation of Penalties and Labeling

CubeCut generates a network $N = ((G = (V(G), E(G))), c, s, t)$, where G is a directed, two-terminal graph and $|V(G)| = |P'| + 2$. Each vertex $v \in V(G)\backslash\{s,t\}$ corresponds to exactly one voxel $p \in P'$ and no two vertices correspond to the same voxel. In the following, v_p will denote a mapping of the vertex $v \in V(G)\backslash\{s,t\}$ onto its corresponding voxel $p \in P'$ and p_v will describe the reverse mapping. The source s and the sink t have no counterparts in P' and thus they are referred to as *virtual* nodes [23].

In $E(G)$, there exist two types of edges [21]:

- *i-links* (**i**nter-links) connect vertices $v \in V(G)\backslash\{s,t\}$ with each other. The i-links are further subdivided into *z-edges* and *xy-edges*, where z-edges connect vertices corresponding to neighboring voxels of the same

ray (e.g. $(v_{i_n}, v_{(i+1)_n})$), while xy-edges connect vertices corresponding to voxels of different rays (e.g. (v_{i_n}, v_{j_m})).

- *o-links* (**o**utward-links) connect all vertices $v \in V(G)\backslash\{s,t\}$ with the source s (*s-links*) and the sink t (*t-links*). Hence, there are two o-links for each vertex.

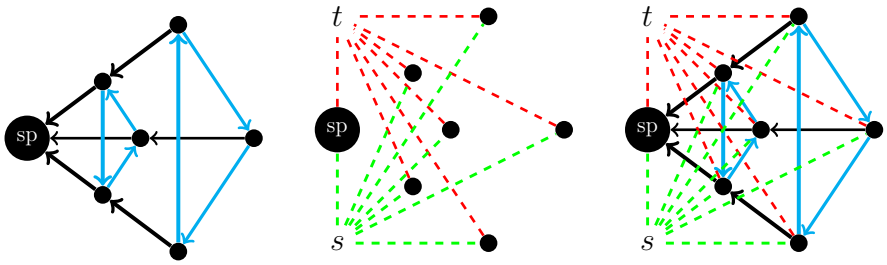


Figure 21: Illustration of the different kinds of edges. (a) i-links: z-edges (black), xy-edges(blue). (b) o-links: s-links(green), t-links(red). (c) whole graph.

The capacities of the i and o-links reflect the penalty functions D and V in the following manner [21]:

$$\forall v \in V(G)\backslash\{s,t\} : c(s,v) = D_p((t(v_p) = L_t)),$$
$$\forall v \in V(G)\backslash\{s,t\} : c(v,t) = D_p((t(v_p) = L_s)),$$
$$\forall (v,v') \in E(G) : c(v,v') = V_{p,p'}(t(v_p), t(v'_p)).$$

Note that the skew symmetry constraint does not have an effect since by convention $c(v,v') = 0$ is assumed, if $(v,v') \notin E(G)$ (compare Ch. 4.3.1).

After the graph has been set up, CubeCut determines a minimal s-t-cut (S,T) by deploying the Boykov-Kolmogorov algorithm[9] and then it labels P' as follows:

$$\forall v_p \in P' : t(v_p) = \begin{cases} L_s & \text{if } v \in S; \\ L_t & \text{else.} \end{cases}$$

Since by definition, the capacity of a minimal s-t-cut is minimal among all possible s-t-cuts (see Ch. 4.3.3), the labeling above minimizes (2).

[9] http://vision.csd.uwo.ca/code/, accessed: 07/20/2012.

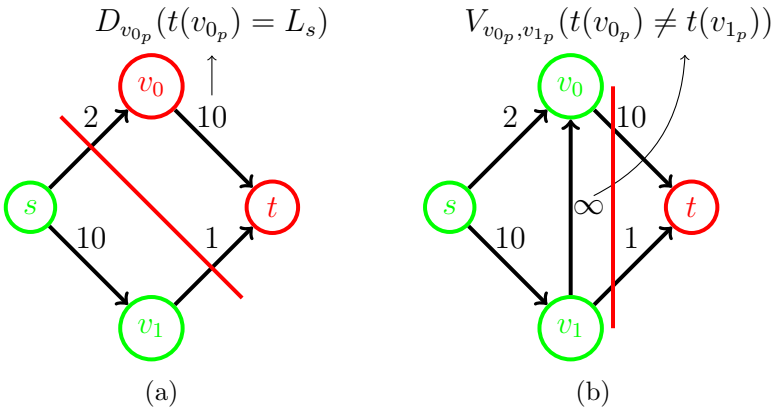


Figure 22: Illustration of the penalty effect. (a) shows a network without i-links. (b) shows a network with an i-link. The red line depicts a minimal cut.

6.4 Z-Edges: Onetime Cut per Ray

Since each ray intersects with the outer boundaries of the vertebral body only once, a set of *z-edges* is introduced that ensures that each ray is exactly cut one time by a minimal s-t-cut [29]:

$$A_z = \{(v_{i_r}, v_{i-1_r}) | 1 < i \leq k \wedge 1 \leq r \leq n\},$$

where n is the user-defined number of rays and k the total number of voxels per ray, again[10].

The set of z-edges connects each vertex v_i with its predecessor v_{i-1} on the same ray (compare figure 23 (a) and (b)). The capacities of all z-edges are initialized to ∞. Therefore, it costs ∞ each time a z-edge is cut.

By making sure that the seed point is in S and that the last voxel on each ray is in T (see next section), a minimal s-t-cut (S, T) has to cut each ray at least once. Yet, it does not cut any ray more than once because that would cost at least $2 \cdot \infty$. This is why a ray is cut exactly one time. The next section explains how CubeCut encourages this cut to happen close in front of the vertebral body's outer boundaries.

[10]In what follows, v_{i_r} will denote the corresponding vertex of the i^{th} voxel on ray r. Furthermore, if only one ray is being discussed, the indexing r might be omitted.

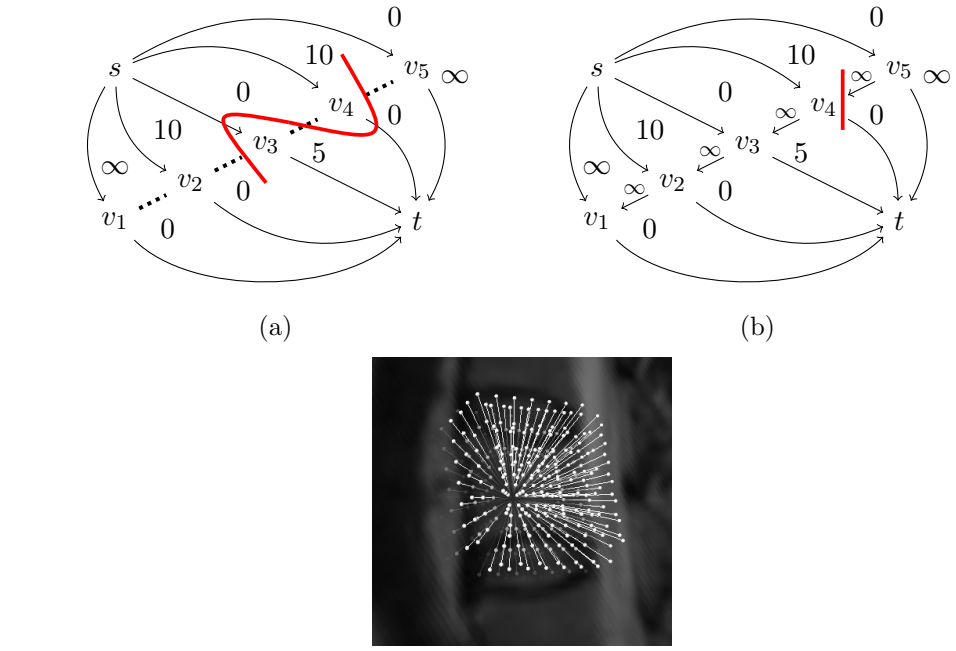


Figure 23: Illustration of the z-edges principle. (a) shows a ray without z-edges: The minimal s-t-cut (red) cuts the ray twice with a capacity of 0. (b) shows the same ray with z-edges. The ray is only cut once. The capacity of the minimal s-t-cut is $\infty + 5$. (c) shows z-edges, embedded into an MR image.

6.5 O-links: Marking the Outer Boundaries

6.5.1 Frames of Reference

A voxel p_{i_r} is characterized by $(x_{i_r}, y_{i_r}, z_{i_r}, g_{i_r})$ where $x_{i_r}, y_{i_r}, z_{i_r} \in \mathbb{N}_0$ denote the voxel's position in the image and $g_{i_r} \in \mathbb{R}_{\geq 0}$ denotes its grey value[11]. CubeCut investigates a small cube $((x_1, y_1, z_1), (x_2, y_2, z_2))$ around the user-defined seed point (inside the vertebra) and determines its interval of grey values

$$I = [min(GV), max(GV)],$$

[11]Simplified. Voxel coordinates assumed.

where

$$GV = \{\pi_4(x, y, z, g) | x_1 \leq x \leq x_2, y_1 \leq y \leq y_2, z_1 \leq z \leq z_2\}$$

is the multi set of all grey values within the cube and $\pi_i(\cdot)$ is a projection onto the i^{th} element of a tuple. Furthermore, CubeCut also iterates over the cube to determine an average grey value g_{avg} by

$$g_{avg} = \frac{1}{|GV|} \cdot \int_{x_1}^{x_2} \int_{y_1}^{y_2} \int_{z_1}^{z_2} \pi_4(x, y, z, g) dx\ dy\ dz.$$

In the course of weighting the o-links, the interval I and the average grey value g_{avg} are used as frames of reference.

6.5.2 Capacities

Figure 24 depicts the fundamental principle of how CubeCut assigns capacities to the o-links.

The ∞-weighting in line 0 ensures that the seed point is tagged with L_s. The premise on which this is based is that the user defines the seed point within the vertebral body (in the center).

Furthermore, it is assumed that the last voxel on each ray (p_{k_r}) lies outside the vertebra (since the user is supposed to define a ray length that exceeds the vertebral body). So as to make sure that the last voxels are tagged with L_t, the t-links (v_{k_r}, t) are also ∞-weighted while for all rays $c(s, v_{k_r})$ is consequently initialized to zero (line 4 - 6).

The capacities of all of the other, intermediate o-links reflect the value difference between a voxel and its predecessor on the ray (lines 8,9,11 & 12). This is in order to "mark" the outer boundaries:

As already mentioned above, the rays expand from the user-defined seed point in the center of the vertebral body and they eventually intersect with the outer boundaries. Ignoring occasional outliers and homogeneous object-background transition regions for now, the inner vertebral body is characterized by a homogeneous set of voxel grey values, which are all higher or lower than the grey values that make up the outer boundaries (e.g. cortical bone, spinal canal, compare Ch. 2.1).

Thus, on each ray, the difference in value between the last voxel in the vertebral body and the first voxel on the outer boundaries can be assumed

```
0   c(s, v_1) ← ∞
1   c(v_1, t) ← 0
2   assign(ray r)
3   ∀p_{i_r} = (x_i, y_i, z_i, g_i) ∈ r\{p_{1_r}}
4       if  i == k
5         c(s, v_{i_r}) ← 0
6         c(v_{i_r}, t) ← ∞
7       else if  g_i ∈ I  or  abs(g_avg − g_i) ≤ abs(g_avg − g_{i−1})
8           c(s, v_{i_r}) ← abs(abs(g_avg − g_i) − abs(g_avg − g_{i−1}))
9           c(v_{i_r}, t) ← 0
10       else
11          c(s, v_{i_r}) ← 0
12          c(v_{i_r}, t) ← abs(abs(g_avg − g_i) − abs(g_avg − g_{i−1}))
```

Figure 24: Pseudo code of s-t-weighting. Each ray r consists of k voxels.

high. Taking the condition in line 7 into account, the outer boundaries therefore implement high t-link capacities (line 12). Note that this makes a cut right in front of the corresponding vertices very probable. The next sections explain how peculiarities and anomalies in vertebral MRI data sometimes prevent a cut from happening right in front of the outer boundaries and how CubeCut addresses these adverse effects.

6.6 Adverse Effects on the Segmentation Result

A cut right in front of the outer boundaries is a cut that separates the last vertex that corresponds to a voxel which is still located inside the vertebral body from the subsequent ones on the same ray. If, for each ray, the cut takes place right in front of the outer boundaries of the vertebral body, then CubeCut returns a satisfactory segmentation result.

Let v_{i_r} be the first vertex on a ray r that corresponds to a voxel on the

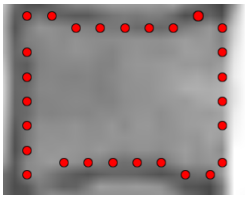


Figure 25: Successful cut.

outer boundaries/background. If a minimal s-t-cut (S, T) cuts the ray right in front of v_{i_r}, so that $v_{(i-1)_r} \in S$ and $v_{i_r} \in T$, then

$$\sum_{j=i}^{k-1} c(s, v_{j_r}) < \sum_{j=i}^{k-1} c(v_{j_r}, t) \tag{3}$$

and

$$\forall h < i : h > 1 \Rightarrow \sum_{j=h}^{i-1} c(v_{j_r}, t) \leq \sum_{j=h}^{i-1} c(s, v_{j_r}). \tag{4}$$

For most rays, the two (minimum) conditions hold true and thus, the cut takes place right in front of the outer boundaries.

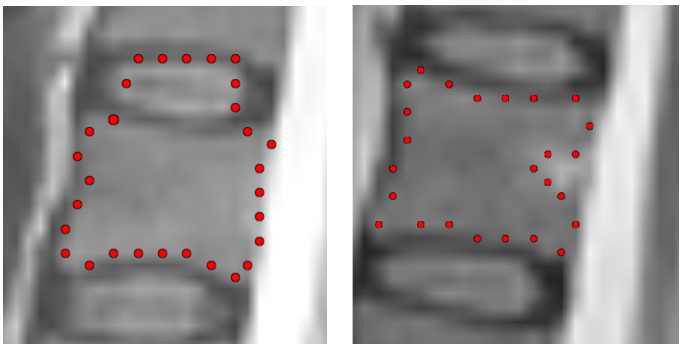
(3) usually holds true because the value difference between p_{i_r} and $p_{(i-1)_r}$ is greater than the sum of the subsequent s-weights since behind the outer boundaries, the rays mostly penetrate homogeneous areas dissimilar from the vertebral body (compare Figure 24 lines 7 & 8). (4) holds true for most rays because of the homogeneity of voxel grey values in the vertebral body and their similarity to the close environment of the seed point (compare Figure 24, lines 7,8 & 12). Nevertheless, there are exceptions.

Figure 26 depicts such exceptions. (a) clearly shows a 2-dimensional view of a segmentation result that overruns the vertebral body in the upper part. For the corresponding rays, (3) does not hold true:

The similarity between the vertebral and the intervertebral voxels, in terms of their grey values, can easily be recognised. Furthermore there are minor variations of grey values in the intervertebral disc. As a consequence, the condition in Figure 24, line 7 holds true for a sufficient number of background voxels on each of the affected rays, which is why (3) is not satisfied.

Thus, the overrun occurs. Observe that the same applies to homogeneous object-background transition regions (compare Ch. 2.1). Among others, CubeCut tackles this problem by introducing a coefficient w, which loads the s-weights according to their distance from the seed point (see next section). The pseudo code in Figure 24 (line 8) is extended to:

$$\mathbf{c}(\mathbf{s}, \mathbf{v_{i_r}}) \leftarrow \mathbf{w}(\mathbf{i}, \mathbf{k}) \cdot \mathbf{abs}(\mathbf{abs}(\mathbf{g_{avg}} - \mathbf{g_i}) - \mathbf{abs}(\mathbf{g_{avg}} - \mathbf{g_{i-1}}))$$



(a) (b)

Figure 26: Adverse effects on segmentation results (2-dimensional view). (a) shows an overrun in the upper part due to a violation of condition (3). (b) shows a segmentation result affected by an outlier which causes a violation of condition (4). The cut happens too close to the seed point (not shown) in the middle of the vertebra because there is a light area similar to the spinal canal.

Another phenomenon that negatively affects the segmentation result is outliers. *Outliers* share all relevant properties (grey values) that distinguish the vertebra's boundaries except that they are part of the inner vertebral body.

To be specific, an outlier causes the violation of (4). On the corresponding ray, the cut then happens too close to the seed point (see Figure 26(b)). CubeCut decreases the possible adverse effects due to outliers by imposing a smoothness constraint on the segmentation result. In addition, the smoothness constraint also addresses the problem of a violation of (3), as discussed above. The next two sections present CubeCut's problem-solving

approaches in detail. The first matter to be addressed will be the loading of the s-capacities and then the smoothness constraint will be discussed.

6.7 Loading the s-Capacities

The coefficient $w(\cdot)$ loads an s-capacity according to the corresponding voxel's (p_{i_r}) position on the ray. For a ray r, consisting of $k(> 1)$ voxels, it is defined as:

$$w(i, k) = mi + b,$$

where

$$k \geq i \in \mathbb{N}_{>0}$$

and

$$m = -\frac{1}{k-1}$$

and

$$b = 1 - m.$$

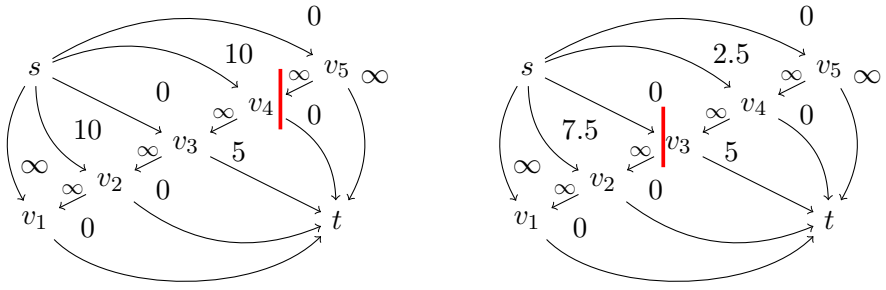


Figure 27: Effect of the coefficient w. In (b), w is applied on the s-weights in (a): The cut, with a capacity of $\infty + 2.5$, now happens closer to the seed point. Note that the same cut in (b) would have cost $\infty + 10$ whereas the cut depicted in (b) has a capacity of only $\infty + 5$.

Observe that since the voxels are distributed uniformly on each ray, $w(i, k) = 1$ for the seed point $(p_{i=1})$, $w(\cdot, k) = 0.5$ for a voxel that is half way

on a ray (see Figure 28) and $w(\cdot, k) = 0$ for the last voxel on each ray. A voxel far away from the seed point is more likely to be outside the vertebral body.

CubeCut takes this into account by decreasing its s-capacity accordingly, thereby reducing the risk of a cut being located behind the outer boundaries of the vertebral body because

$$\sum_{j=i}^{k-1} c(s, v_{j_r}) > \sum_{j=i}^{k-1} w(j, k) \cdot c(s, v_{j_r}).$$

As already mentioned above, the coefficient is not the only measure CubeCut takes in order to counteract a violation of (3): The smoothness constraint, which also addresses a violation of (4), will be the subject matter in the next section.

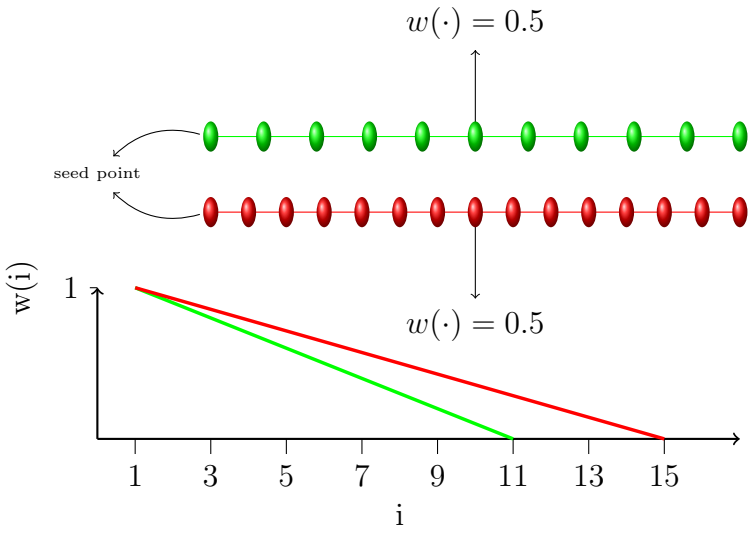


Figure 28: Courses of $w(i, 11)$ (green) and $w(i, 15)$ (red). The upper part illustrates that $w(i, k)$ reflects the position of the voxel p_i on a ray consisting of k uniformly distributed voxels. Note that that w is only partially defined for the natural numbers but that CubeCut never calls w with an argument in the undefined scope (compare Figure 24).

6.8 XY-edges: Imposing a Smoothness Constraint

The smoothness constraint is based on the optimal surface segmentation algorithm developed by Kang Li et al. [29]. It is useful to first discuss it conceptually, slightly detached from the context of vertebral segmentation.

A single, feasible surface in a volumetric Image $I = (X, Y, Z)$, where $X, Y, Z \subset \mathbb{N}_0$, can be characterised by a bijection

$$S : XY \to Z,$$

where $XY \subseteq X \times Y$ is a cohesive area. Li et al. refer to a surface as *feasible* if two smoothness constraints are satisfied:

$$\forall (x, y), (x + 1, y) \in XY : |S(x, y) - S(x + 1, y)| \leq \Delta_x$$

and

$$\forall (x, y), (x, y + 1) \in XY : |S(x, y) - S(x, y + 1)| \leq \Delta_y.$$

Δ_x and Δ_y constrain the degree to which the surface "moves" upwards or downwards in x- or y-direction within an interval of one: $S(x, y)$ and $S(x + 1, y)$ as well as $S(x, y)$ and $S(x, y + 1)$ are neighboring x- and y-positions. Thus, two neighbors on a feasible surface cannot be arbitrarily distant from each other. Hereby, the smoothness constraints assure what Li et al. refer to as "surface connectivity". Observe that for a plane $\Delta_x = \Delta_y = 0$.

Now consider a number of equidistant rays that consist of the same number of uniformly spread voxels and which all extend parallel to the z-axis. The voxels that make up a ray do not necessarily have to lie on neighboring positions in the image. Furthermore, for convenience, assume that I is a binary image with only two possible values for each voxel: *colored* xor *white*. In addition, let all rays intersect with a colored surface $S(XY)$ in I, which means that each ray extends from XY and shares exactly one colored voxel with the surface.

In this context, in which only a subset of the image's voxels is observed, the smoothness constraint has to be defined via the neighborhood relations of the rays. Figure 29 shows four neighboring rays in x-direction (same y-value for each voxel), which extend parallel to the z-axis, as described above. Here, a smoothness parameter $\Delta_x = 1$ means that for a colored voxel that is considered part of the surface, all voxels on adjacent rays that are also classed with the surface voxels must lie on the same "z-layer" or the next upper or lower one. An outlier in this context is a colored voxel that exceeds the prescribed maximum distance.

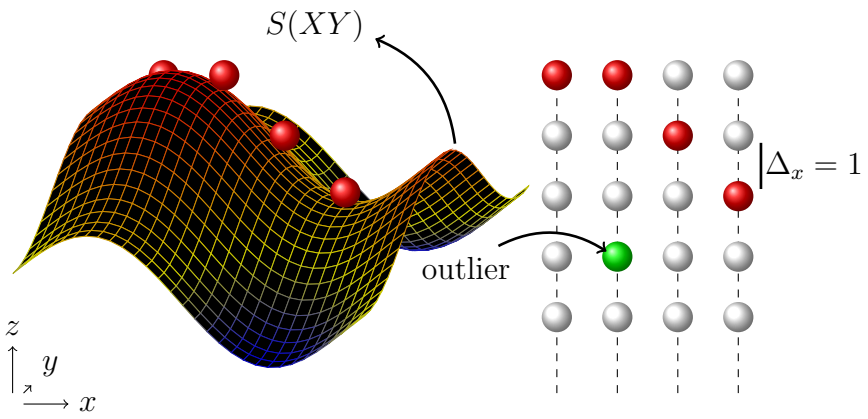


Figure 29: A feasible surface and intersecting rays (transformed in x-direction for a better visibility). The green node depicts an outlier as it would violate the smoothness constraint $\Delta_x = 1$ if classed with the surface voxels.

CubeCut allows the user to impose a smoothness constraint on the segmentation result. It interprets each of the six sides of a vertebral body's outer boundaries (from a sagittal view: front, back, top, bottom, right, and left) as a feasible surface. Furthermore, it takes into account that the six surfaces are anatomically connected, which is why the neighborhood relations overlap at the "edges" of the boundaries (see Figure 32).

6.8.1 Implementation

CubeCut implements the smoothness constraint $\Delta \in \mathbb{N}_0$ by introducing a set of infinity-weighted xy-edges

$$A_{xy} = \{(v_{i_r}, v_{(max\{i-\Delta,1\})_{r'}})|(r, r') \in N_4\}.$$

N_4 denotes a 4−neighborhood as illustrated in Figure 30. As already mentioned above, the neighborhood relations overlap at the "edges" of the cubic voxel subset that the algorithm observes (see Figure 32).

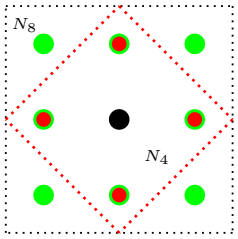


Figure 30: Illustration of 4-neighborhood (N_4, red square) and 8-neighborhood (N_8, black square).

The infinity-weighting of the xy-edges ensures that a minimal s-t-cut cuts the rays in a way such that the vertebra is segmented within the boundaries of the user-defined smoothness constraint Δ. Note that for a given Δ-value, an 8−neighborhood would increase the "stiffness" of the segmentation result.

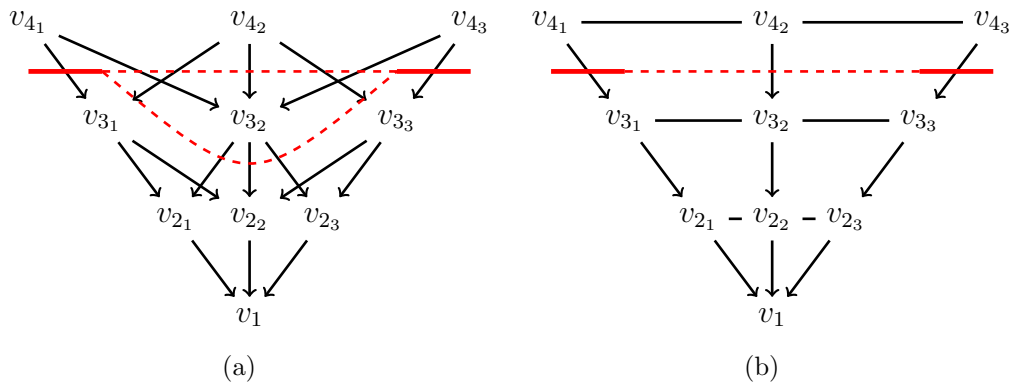
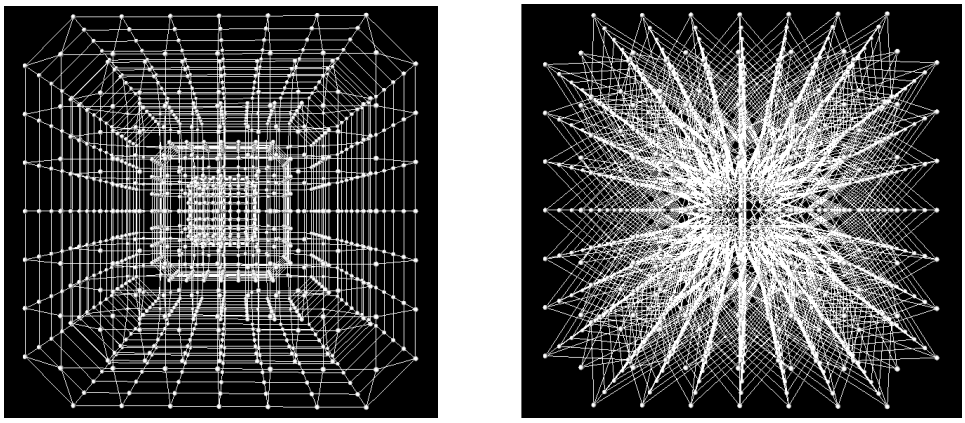


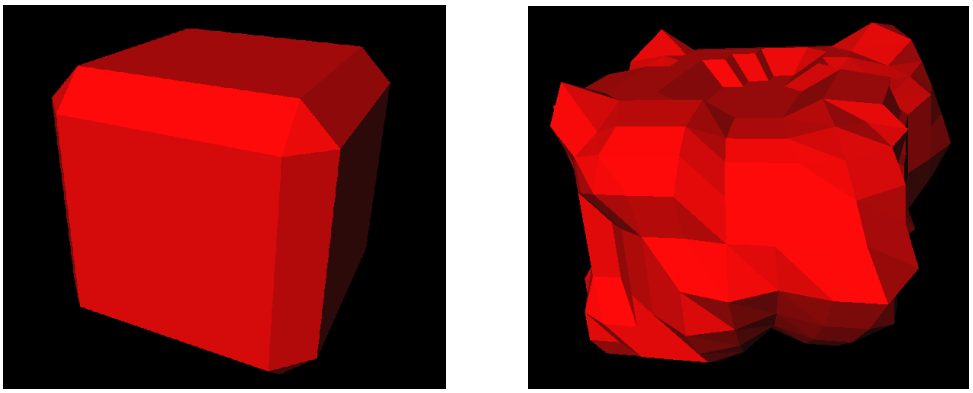
Figure 31: Illustration of the xy-edges principle. (a) shows a minimal cut (thick lines) and the two possible continuations (dashed lines) within the boundaries of a smoothness constraint $\Delta = 1$. All other cuts would have a capacity greater than $7 \cdot \infty$. (b) shows the only possible continuation within the boundaries of a Δ-value of 0, where the cut has a capacity of $3 \cdot \infty$.



(a) (b)

Figure 32: Topology of xy-edges for $\Delta = 0$ (a) and $\Delta = 1$ (b).

For a smoothness constraint $\Delta = 0$, any minimal s-t-cut results in a regular, cubic segmentation result, whereas a Δ-value greater zero allows a corresponding deviation. The figure above shows the topology of the xy-edges for a Δ-value of zero and a Δ-value of one. Below, corresponding segmentation results are depicted.



(a) (b)

Figure 33: Segmentation results for $\Delta = 0$ (a) and $\Delta = 1$ (b).

7 Evaluation

7.1 Results

A C++ implementation of the new segmentation algorithm CubeCut was evaluated within the medical image processing platform MeVisLab [1] version 2.2.1 on a 2.1 GHz x64-based PC with 4 GB RAM running the Microsoft Windows 7 Home Premium (SP1) operating system, version 6.1.7601. Computationally, the most intensive parameter settings resulted in a maximum processing time of 19.4 seconds (for subset selection, graph construction, s-t-cut and triangulation) and a comparison with manually segmented vertebrae resulted in a mean DSC of 81.88%, ranging from 71.64% to 86.69%.

After the s-t-cut (S, T), the surface of CubeCut's segmentation result was determined by a triangulation over the last voxels in S on a ray, using the MeVisLab C++ class `SoIndexedFaceSet`. For the voxelization of the data thus obtained, the MeVisLab module `VoxelizeInventorScene` was deployed[12]. The manual segmentations were conducted by a physician in a slice-by-slice manner. For the voxelization of the manually segmented slices, `CSOConvertTo3DMask` was used.

Table 4 summarizes the results and statistics of the vertebral segmentations. The five lumbar vertebrae (L1 - L5) and a thoracic one (T11) were segmented both manually (by the physician) and automatically (by CubeCut). Since the parameter settings in CubeCut are essential for the outcome and processing time, the subsequent tables list the corresponding settings.

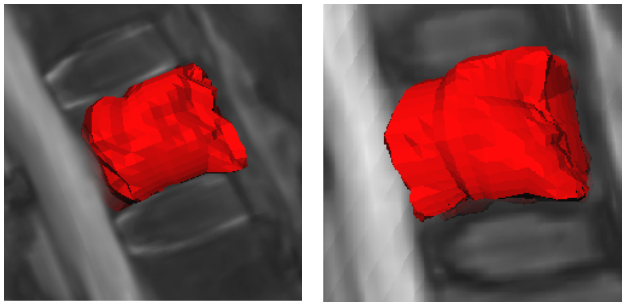
Table 4: Segmentation and comparison statistics: volumes of manual (m.) and automatic (a.) segmentation results, volume of overlaps (o.) as well as numbers of voxels (vox.) and dice similarity coefficient (dsc).

#	m./mm^3	a./mm^3	m.vox.	a.vox	o./mm^3	o.vox.	dsc(%)
L1	23860.6	26314.3	2927	3228	21749.3	2668	86.6937
L2	27423	27431.1	3364	3365	23068.2	2832	84.173
L3	33830.4	28776.2	4150	3530	25686.6	3151	82.0573
L4	27121.4	23901.4	3327	2932	21064.5	2584	82.5691
L5	22165	17795.4	2719	2138	14314.7	1756	71.6442
T11	15423.4	16638	1892	2041	13491.4	1655	84.1597

[12]For replication, all parameter settings of all relevant modules are listed in the appendix.

Table 5: CubeCut parameter settings and corresponding processing times. ppr stands for *points per ray*, ⍉ denotes the edge length of the outer cube (biggest) measured in voxels (v.) and centimeters, dist denotes the distance range between voxels on the smallest cube and the biggest respectively, Δ is the smoothness constraint, spt is the position of the user-defined seed point in the image and t denotes the corresponding overall processing time.

#	ppr	⍉/vox.	⍉/cm	dst/cm	Δ	spt	t/sec
L1	40	40	8	0.016-0.68	4	(73.5,44.5,18.5)	19.1
L2	40	40	8	0.016-0.68	4	(70.5,62.5,18.5)	19.0
L3	40	40	8	0.016-0.68	4	(68.5,80.5,18.5)	19.2
L4	40	40	8	0.016-0.68	4	(67.5,99.5,18.5)	19.4
L5	8	24	4.8	0.025-0.6	1	(70.5,117.5,18.5)	1.3
T11	40	40	8	0.016-0.68	4	(80.5,12.5,18.5)	19.4



(a) (b)

Figure 34: 3D segmentation results.

7.2 Discussion

According to recent publications in the field of vertebral segmentation in MRI data, an initial mean DSC of around 80% can be considered "promising" [30]. However, the algorithm (CubeCut) is still in need of improvement. Furthermore, in order to draw a final conclusion on solid ground, more extensive validations are needed.

Findings show that it takes a trained physician an average of 10.75 ± 6.65 minutes to segment a single vertebra slice by slice [23]. In terms of processing time, CubeCut therefore outperforms its target group by 3.78 - 10.43 minutes.

Furthermore, it accurately detects a gradually increasing volume from T11 to L3 and the patient's peculiar and unusually low volume of L4 as well as the anatomically regular lower volume of L5 (compare Table 4). However, a visual evaluation/comparison of CubeCut's segmentation results indicates that the algorithm frequently segments the same specific areas of a vertebral body inaccurately. Recognizing the cubic shape of a vertebral body, on sagittal slices, these areas could be referred to as the vertebral body's "vertices" (see Figure 35).

Future versions of CubeCut could overcome this problem by a densification of rays in the corresponding spaces or by allowing the user to adjust the segmentation result manually. In addition, the present version of CubeCut already allows an arbitrary increase of the s-t-network's complexity (precision), at the expense of processing time, of course.

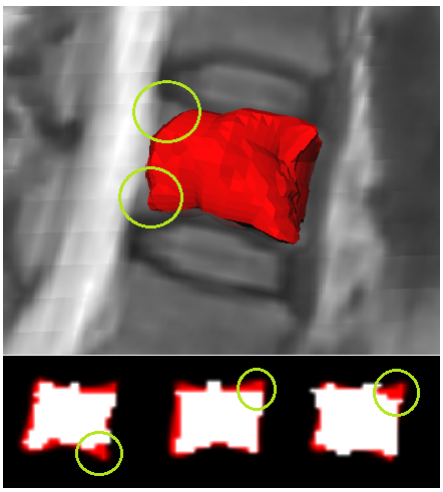


Figure 35: Failed segmentations. "Vertices" of the vertebral body's outer boundaries were not detected (green circles). The upper part shows a 3D segmentation result, the lower part shows the 2D overlaps of manual (red) and automatic (white) segmentation results.

The accuracy of the algorithm's segmentations depends on the parameter settings as well as the position of the user-defined seed-point. Nevertheless, the results suggest that once appropriate parameter settings are found for a vertebral body, these settings can also be successfully applied on other vertebrae (compare Table 4). L5 is an exception but note that L5 also differs significantly from the other lumbar and thoracic vertebrae in terms of bone morphology, shape and size.

In the course of evaluation, overruns still occurred. If, however, the smoothness constraint and the loading of the s-weights were disabled, the quantity of overruns and adverse effects due to outliers increased significantly.

8 Conclusion

A novel 3D vertebral body segmentation approach was presented. Although there is still room for improvement, the graph-based algorithm (CubeCut) achieved a promising average DSC value of 81.88% in an initial evaluation of a C++ implementation within the medical processing platform MeVisLab. Furthermore, its suitability for medical use, in terms of processing time, could be demonstrated.

Enhancing the previously introduced 2D strategy SquareCut, CubeCut recognizes the cubic shapes of vertebral bodies. It allows the user to impose a smoothness constraint on the segmentation result which determines its degree of deviation from a cubic-shaped template. As a result, it effectively reduces adverse effects due to weak boundaries (homogeneous object-background transition regions) and outliers, for example.

CubeCut generates a two-terminal s-t-network where the vertices correspond to a cubic shaped subset of the image's voxels. The capacities of the terminal edges reflect a voxel's affiliation with the object (vertebral body) and the background, while the topology of non-terminal ∞-weighted edges implements the smoothness constraint. After network construction, a minimal s-t-cut is computed which determines the segmentation result.

For the mincut computation, CubeCut deploys the Boykov-Kolmogorov algorithm. This is based on experimental findings which suggest that despite a worst case complexity of $O(v(G)^2 e(G)|C_{min}|)$, where $v(G)$ is the number of vertices, $e(G)$ the number of edges and $|C_{min}|$ the capacity of a minimal cut, the algorithm is most effective for networks of low complexity. CubeCut implements an N_8 neighborhood system (including terminal edges) which means the network can be classed with the less complex ones.

The graph construction takes CubeCut $O(v(G) + 8 \cdot v(G))$ steps and this results in an overall worst time complexity of $O(v(G)^2 e(G)|C_{min}|)$. The parameter settings that resulted in a DSC value of 86.6937% took CubeCut 19.1 seconds to terminate (graph construction, mincut computation and triangulation). It was found that a slice-by-slice segmentation of a vertebra took trained physicians 10 ± 6.65 minutes on average and a susequent conversion into a 3D mask is also still needed here.

References

[1] *MeVisLab Reference Manual*, MeVis Medical Solutions AG and Fraunhofer MEVIS, September 20 2012. [Online]. Available: http://www.mevislab.de/

[2] M. Y.M.Chen, T. L. Pope, and D. J. Ott, Eds., *Basic Radiology*, 2nd ed. McGraw Hill Medical, 2011.

[3] R. L. Drake, A. W. Vogl, and A. W. M. Mitchell, Eds., *Gray's Anatomy for Students*, 2nd ed. Richardson - Elsevier/Churchill Livingstone, 2010.

[4] A. F. Joaquim, C. A. Sansur, D. K. Hamilton, and C. I. Shaffrey, "Degenerative lumbar stenosis," *Arquivos de Neuro-Psiquiatria*, vol. 67(2B), pp. 553–558, 2009. [Online]. Available: http://www.scielo.br/

[5] J. N. Weinstein, J. D. Lurie, T. D. Tosteson, B. Hanscom, A. N. Tosteson, E. A. Blood, N. J. Birkmeyer, A. S. Hilibrand, H. Herkowitz, F. P. Cammisa, T. J. Albert, S. E. Emery, L. G. Lenke, W. A. Abdu, M. Longley, T. J. Errico, and S. S. Hu, "Surgical versus nonsurgical treatment for lumbar degenerative spondylolisthesis," *The New England Journal of Medicine*, vol. 356, pp. 2257–2270, 2007. [Online]. Available: http://www.nejm.org/

[6] J. Miao, S. Wang, Z. Wan, W. Park, Q. Xia, K. Wood, and G. Li, "Motion characteristics of the vertebral segments with lumbar degenerative spondylolisthesis in elderly patients," *European Spine Journal*, 2012, epub ahead of print. [Online]. Available: http://www.ncbi.nlm.nih.gov/

[7] K. H. Zou, S. K. Warfield, A. Bharatha, C. M. C. Tempany, M. R. Kaus, S. J. Haker, W. M. Wells, F. A. Jolesz, and R. Kikinis, "Statistical validation of image segmentation quality based on a spatial overlap index," *Academic Radiology*, vol. 11(2), pp. 178–189, 2004. [Online]. Available: www.academicradiology.org/

[8] K. G. Baum, E. Schreyer, S. Totterman, J. Farber, J. Tamez-Pena, and P. Gonzalez, "Application of the dice similarity coefficient (dsc) for failure detection of a fully-automated atlas based knee mri segmentation method." ISMRM Annual Meeting, March 2010, stockholm, SW. [Online]. Available: http://www.qmetricstech.com/

[9] L. R. Dice, "Measures of the amount of ecologic association between species," *Ecology*, vol. 26, pp. 297–302, 1945.

[10] D. Jungnickel, *Graphs, Networks and Algorithms*, 5th ed. Springer, 2004.

[11] J. A. Bondy and U. S. R. Murty, *Graph Theory*. Springer, 2008.

[12] U. Knauer, *Algebraic Graph Theory : Morphisms, Monoids, and Matrices*. De Gruyter, 2011.

[13] A. Gibbons, *Algorithmic Graph Theory*. Cambridge University Press, 1985.

[14] H. Walther, *Anwendungen der Graphentheorie*. Vieweg+Teubner Verlag, 1979.

[15] N. Christofides, *Graph Theory - An Algorithmic Approach*, 4th ed. Academic Press, 1975.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press and McGraw-Hill, 2001.

[17] L. R. Ford and D. R. Fulkerson, *Flows in Networks*, 6th ed. Princeton University Press, 1974.

[18] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization : Algorithms and Complexity*. Prentice Hall, 1982.

[19] H. Walther and G. Nägler, *Graphen, Algorithmen, Programme*. Springer, 1987.

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

[21] Y. Boykov and V. Kolmogorov, "An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, pp. 1124–1137, 2004.

[22] E. A. Dinic, "Algorithms for solution of a problem of maximum flow in networks with power estimation," *Soviet Math. Dokl.*, vol. 11, pp. 1277–1280, 1970.

[23] J. Egger, T. Kapur, T. Dukatz, M. Kolodziej, D. Zukic, B. Freisleben, and C. Nimsky, "Square-cut: A segmentation algorithm on the basis of a rectangle shape," *PLoS ONE*, vol. 7(2), 2012.

[24] J. Egger, B. Freisleben, C. Nimsky, and T. Kapur, "Template cut: A pattern-based segmentation paradigm," *Nature - Scientific Reports, Nature Publishing Group (NPG)*, vol. 2(420), 2012.

[25] J. Egger, M. H. A. Bauer, D. Kuhnt, B. Carl, C. Kappus, B. Freisleben, and C. Nimsky, "Nugget-cut: A segmentation scheme for spherically- and elliptically-shaped 3d objects," *32nd Annual Symposium Of The German Association For Pattern Recognition (DAGM)*, pp. 383–392, 2010.

[26] J. Egger, B. Freisleben, R. Setser, R. Renapuraar, C. Biermann, and T. O'Donnell, "Aorta segmentation for stent simulation," *12th International Conference On Medical Image Computing And Computer Assisted Intervention (MICCAI), Cardiovascular Interventional Imaging And Biophysical Modelling Workshop*, pp. 1 – 10, 2009.

[27] J. Egger, T. O'Donnell, C. Hofgartner, and B. Freisleben, "Graph-based tracking method for aortic thrombus segmentation," *Proceedings of 4th European Congress For Medical And Biomedical Engineering, Engineering For Health*, pp. 584–587, 2008.

[28] D. Greig, B. Porteous, and A. Seheult, "Exact maximum a posteriori estimation for binary images," *Journal of the Royal Statistical Society, Series B*, vol. 51(2), pp. 271–279, 1989.

[29] K. Li, X. Wu, D. Z. Chen, and M. Sonka, "Optimal surface segmentation in volumetric images - a graph theoretic approach," *IEEE Transactions On Pattern Analysis And Machine Intelligence (PAMI)*, vol. 28, pp. 119–134, 2006.

[30] A. Neubert, J. Fripp, S. Kaikai, O. Salvado, R. Schwarz, L. Lauer, C. Engstrom, and S. Crozier, "Automated 3d segmentation of vertebral bodies

and intervertebral discs from mri," *International Conference on Digital Image Computing Techniques and Applications (DICTA)*, pp. 19–24, 2011.

A Publication

National Center for Image Guided Therapy | BRIGHAM AND WOMEN'S HOSPITAL | HARVARD MEDICAL SCHOOL | Philipps Universität Marburg | Surgical Planning Laboratory

Graph-Based Vertebra Segmentation Using a Cubic Template

Robert Schwarzenberg[a,b], Tina Kapur, Ph.D.[a], William Wells, Ph.D.[a], Christopher Nimsky, M.D., Ph.D.[c], Bernd Freisleben, Ph.D.[b], Jan Egger, Ph.D., Ph.D.[a,b,c]

[a] Dept. of Radiology, Brigham and Women's Hospital, Harvard Medical School, Boston, MA, USA,
[b] Dept. of Mathematics and Computer Science, University of Marburg, Marburg, Germany,
[c] Dept. of Neurosurgery, University of Marburg, Marburg, Germany

{ rs | tkapur | sw | egger }@bwh.harvard.edu, { egger | nimsky }@med.uni-marburg.de, freisleb@informatik.uni-marburg.de

Purpose

The current development of the population's structure leads to a growing part of older patients with a more frequent insistence for surgical treatment like lumbar spinal stenosis (LS), which is the most common cause of spinal surgery in individuals older than 65 years of age [1]. For the assessment of spinal structures such as nerve roots, intervertebral discs and ligamentary constitution, Magnetic Resonance Imaging (MRI) is in general suitable. However, certain changes of the vertebra due to osteoporosis, fractures or osteophytes, require an evaluation of the bone structures via Computed Tomography (CT)-scans, which include radiation exposure [2]. In this contribution, we want to illustrate the capability of MRI-segmentation to reconstruct the vertebral body without x-ray examination, leading to less pre-operative examinations and therefore affecting radiation exposure costs and time-management.

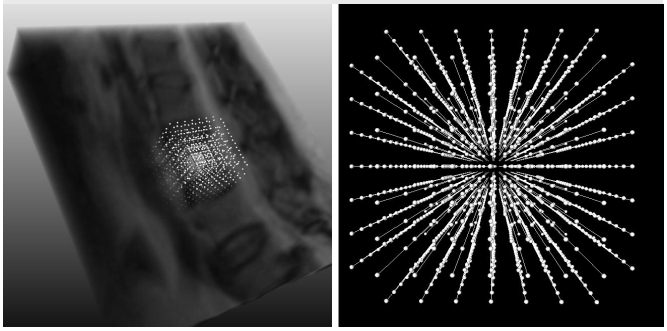


Figure 1: Left: Distribution of vertices. Right: Visualization of z-edges.

Methods

The presented approach is an extension of our previously introduced strategy [3, 4] to a third dimension. It starts by setting up a directed, weighted, two-terminal 3D-graph G = (V,E) (an s-t-network). After its construction, the minimal closed set on the graph is calculated via a polynomial time s-t-cut [5], creating a 3D segmentation of the vertebral body. The vertices $v \in V \setminus \{s,t\}$ are distributed along several rays that extend from a user-defined seed point inside the vertebra and intersect with the vertebral body's outer boundaries. All rays are made up of the same number of vertices and each layer forms a cube shape (see figure 1). There are two types of edges $e \in E$. *n-links* connect all vertices to a virtual source s and a virtual sink t and the n-links' capacities reflect a node's affiliation with either the source (vertebra) or the sink (background). A set of infinity-weighted *i-links* connects the vertices on the rays with each other. The i-links are further subdivided into *z-edges* (see figure 1) and *xy-edges* (see figure 2). The z-edges ensure that each ray is cut exactly one time, while the xy-edges allow the user to impose a smoothness constraint Δ on the segmentation result [6]. A Δ-value of zero results in a regular, cubic shape, whereas a Δ-value greater zero allows a corresponding deviation (see figure 3).

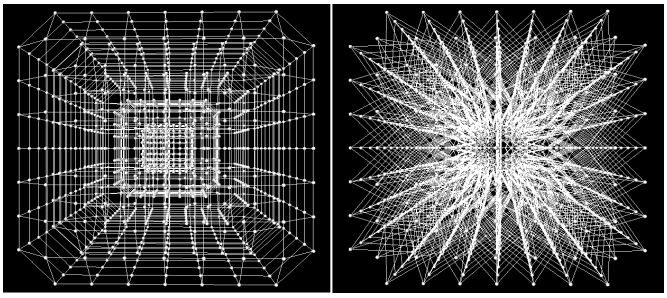


Figure 2: Topology of xy-edges. Left: Smoothness constraint $\Delta = 0$. Right: $\Delta = 1$.

Results

For testing the presented segmentation method we used a C++ implementation within the medical prototyping platform MeVisLab (see http://www.mevislab.de). The overall segmentation – sending rays, graph construction and mincut computation – in our implementation took about twenty seconds on an Intel 2.1 GHz CPU, 4 GB RAM, Windows 7 Home Premium x64 Version, SP 1. We carried out an initial evaluation, segmenting 5 vertebrae: The average DSC was 83%.

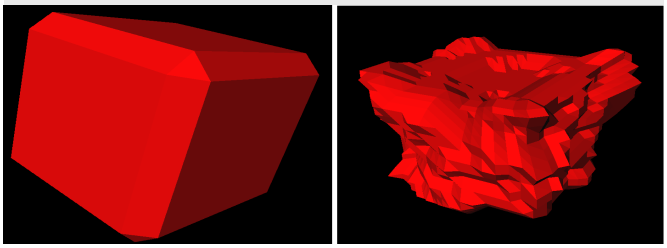


Figure 3: Left: Segmentation result for $\Delta = 0$. Right: Segmentation result for $\Delta = 2$.

Conclusions

In this contribution, we presented the initial results for a novel vertebra 3D segmentation method. The method enhances our recently developed algorithm to a third dimension. Whereas the previously introduced algorithm allowed the calculation of a vertebral area (2D), the method presented here determines the volume of a vertebra (3D) (see figure 4). It constructs an s-t-network within a cubic-shaped template and allows the user to impose a smoothness constraint on the segmentation result which determines the result's deviation from a regular cube shape. The segmentation result is computed by a polynomial s-t-cut, creating an optimal segmentation of the vertebra's outer boundaries. A first evaluation led to an average DSC of 83 %.

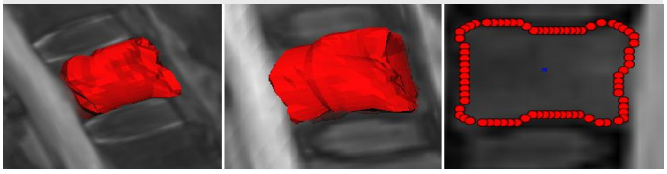


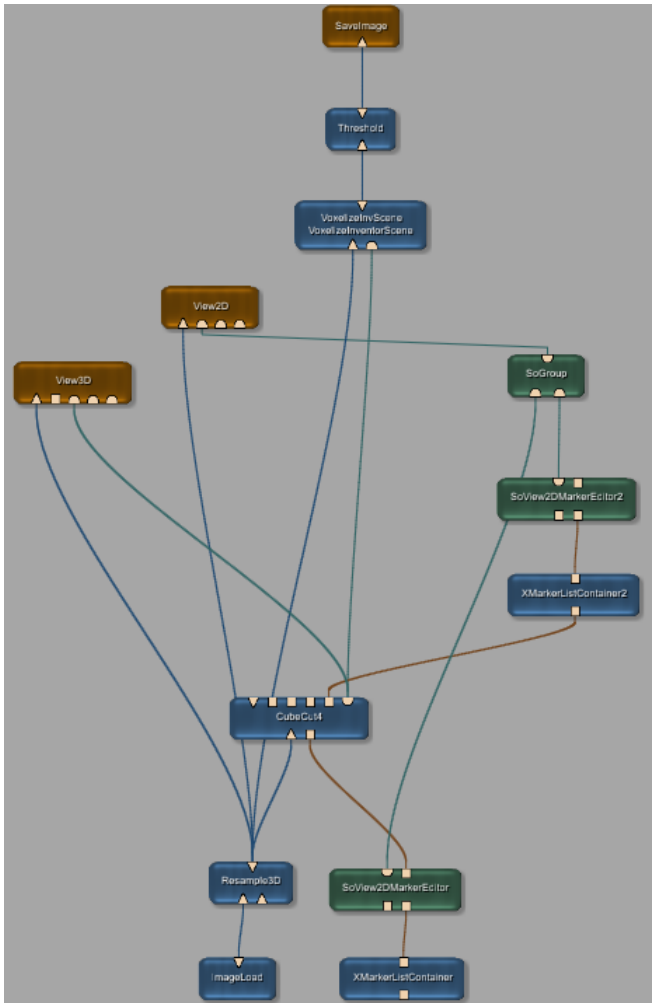
Figure 4: Left and middle: 3D segmentation results. Right: 2D Segmentation result.

References

[1] A.F. Joaquim, et al. Degenerative lumbar stenosis: update. Arq Neuropsiquiatr 67(2B): 553-8, 2009.
[2] P.J. Richards, et al. Spine computed tomography doses and cancer induction. Spine (Phila Pa 1976) 35(4): 430-3, 2010.
[3] J. Egger, T. Kapur, T. Dukatz, M. Kolodziej, D. Zukic, B. Freisleben, and C. Nimsky. Square-Cut: A Segmentation Algorithm on the Basis of a Rectangle Shape. In: PLoS ONE, 2012.
[4] J. Egger, B. Freisleben, C. Nimsky, and T. Kapur. Template-Cut: A Pattern-Based Segmentation Paradigm. In: Nature - Scientific Reports, Nature Publishing Group (NPG), 2(420), 2012.
[5] Y. Boykov and V. Kolmogorov. An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. IEEE Transactions on Pattern Analysis and Machine Intelligence, 26(9), pp. 1124-1137, 2004.
[6] K. Li, X. Wu, D.Z. Chen, and M. Sonka. Optimal Surface Segmentation in Volumetric Images – A Graph-Theoretic Approach. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 28(1): 119-134, 2006.

5th NCIGT and NIH Image Guided Therapy Workshop, September 21, 2012, Joseph B. Martin Conference Center, Boston, MA.

B CubeCut Network


SaveImage
Threshold
VoxelizeInvScene
VoxelizeInventorScene
View2D
View3D
SoGroup
SoView2DMarkerEditor2
XMarkerListContainer2
CubeCut4
Resample3D
SoView2DMarkerEditor
ImageLoad
XMarkerListContainer

C Parameter Settings

Parameter Settings

Dataset: Sch

`Resample3D` Resample Filter: Lanczos 3; Keep Constant: Image size; Use isotropic voxel size ? true; Image Size: x: 159 y: 159 z: 35; Voxel Size: x: 2.01258 y: 2.01259 z: 2.01258; Scale Factor: x: 0.310547 y: 0.310546 z: 2.18625; Voxel Translation: x: 0 y: 0 z: 0; Filtering Tolerance: 0; Filter always ? false;

`VoxelizeInventorScene` Mode: Voxel Distance; Thickness: 0.2; Surface ? true; Anti-alias ? true; Colored ? true; Filled ? true; Include border ? true; User super sampling ? false; Write Voxel Value: 1024; Fill Color: WHITE; Alpha: 1; On ML image change ? false; On Inventor change ? true; On parameter change ? false; Copy input image ? false; DrawStyle as Inventor scene ? false; Triangles ? true; Lines ? false; Points ? false; Output number of collected primitives ? false;

`Threshold` Threshold: 1; Use relative threshold ? false;

`Other` default

D Eigenständigkeitserklärung

Eigenständigeitserklärung

Hiermit versichere ich, dass ich diese Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Stellen meiner Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, habe ich in jedem Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht. Diese Arbeit hat in dieser oder einer ähnlichen Form noch nicht im Rahmen einer Prüfung vorgelegen.

Marburg, den 4. Oktober 2012
